MASTER EN SCIENCES INFORMATIQUES

Etude d'un problème d'intelligence artificielle dans le monde du jeu vidéo l'importance de la pré-production

Noël, Jean-François

Award date:
2011

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'informatique

Année académique 2010-2011

Etude d'un problème d'intelligence artificielle dans le monde du jeu vidéo : l'importance de la pré-production

Jean-François Noël

Mémoire présenté en vue de l'obtention du grade de master en Sciences Informatiques

Promoteur : *Jean-Marie Jacquet*
Maître de stage *: Louis-Marie Rocques*

Résumé

Le présent travail de recherche décrit notre démarche de documentation et de réponse face à problème que nous avons identifié lors de notre expérience au sein de l'industrie de développement de logiciels vidéo-ludiques.

Au cours de notre expérience et au travers de notre documentation, nous avons mis en évidence la tension récurrente entre la phase dite de pré-production et la phase dite de production d'un jeu vidéo.

Nous avons alors cherché à montrer les mérites et qualités de la phase de pré-production au travers de la résolution d'un problème d'intelligence artificielle que nous avons rencontré au cours de notre expérience.

Cette résolution présente donc une définition, conceptualisation et modélisation poussée du problème et des solutions que nous avons proposées. Ces solutions ont ensuite été implémentées et évaluées.

Nous posons alors un regard critique sur la démarche qui a mené à la découverte, à la modélisation et à l'implémentation de ces solutions, afin de mettre en avant les qualités et les défauts d'une telle approche.

Abstract

The research presented here describes our approach to modeling and solving a problem we encountered during our experience as a programmer inside a video game development studio.

Through our experience in said studio and through our readings, we discovered the everlasting tension between the pre-production and the production phases that occurs during the development of a video game.

We then tried to demonstrate the qualities and merits of the pre-production phase through the study of an artificial intelligence problem that we encountered during our experience in the video game studio.

That study describes a throughout definition, design and modeling of the encountered problem and of the solutions we proposed. Those solutions were then implemented and evaluated.

Finally, we go back on the approach and reasoning that led to the discovery, modeling and implementation of those solutions, in order to emphasize the qualities and the flaws of such an approach.

Avant-propos

Ce travail apporte notre participation au débat opposant la phase dite de pré-production et la phase dite de production, au cours du développement d'un logiciel vidéo-ludique. La phase de pré-production concerne la conceptualisation et la modélisation des éléments du jeu, tandis que la production consiste en l'implémentation de ces éléments.

De fait, au cœur de l'industrie du jeu vidéo, deux courants s'opposent. Le premier met en avant les résultats rapides qu'offre une phase de production précoce, tandis que le deuxième oppose la meilleure qualité et documentation offerte par une phase de pré-production poussée.

Par la présente recherche, nous espérons mettre en évidence les qualités d'une pré-production aboutie. Nous n'avons aucunement la prétention de résoudre ce débat, ni de prouver formellement l'ascendant d'une phase sur une autre. Nous cherchons simplement à décrire l'intuition, la réflexion et la recherche qui nous ont conduits en tant que technicien à apprécier une phase de pré-production poussée.

Nous espérons ainsi présenter au lecteur une piste de réflexion qui l'aidera à mieux apprécier le débat éternel entre ces deux phases critiques du développement d'un jeu vidéo.

Nous tenons à remercier notre promoteur, Mr. Jean-Marie Jacquet, pour sa patience et son suivi de notre travail. Nous remercions également Mr. Louis-Marie Rocques pour son aide et sa permission d'utiliser son logiciel vidéo-ludique, *Rulers of Nations*, pour les besoins de ce travail.

Nous remercions de même l'ensemble du personnel du studio *Eversim* pour son accueil lors de notre expérience au sein de leur équipe de développement. Enfin, nous remercions notre famille et nos amis pour leur soutien durant la rédaction de cette recherche.

Table des matières

Glossaire

Asset : Au sens informatique : ressource informationnelle pour un logiciel.

Contrôleur IA : Programme informatique qui régit le comportant d'un *NPC* ou d'un groupe de *NPC* au sein d'un jeu vidéo. C'est lui qui analyse l'environnement des *NPC* et prend des décisions en accord avec ces informations.

E3 : L'*Electronic Entertainment Expo*, ou E3, est une convention annuelle de l'industrie du jeu-vidéo qui se tient aux Etats-Unis et qui rassemble chaque année des dizaines de milliers de visiteurs autour de centaines de stands.

Frames : Une animation, comme une cinématique, se décompose en une séquence d'images jouée à une vitesse particulière. Les images composant cette séquence se nomment des *frames*.

Game Design : Ensemble des éléments de conception d'un jeu qui concernent son contenu, ses règles, son environnement, son scénario, et ses personnages. Le ou les responsables de cette conception sont nommés les *game designers*.

Gameplay : Le *gameplay* d'un jeu reprend tous les aspects interactifs de sa conception.

HUD : *Heads-up Display* ou « affichage tête haute », un composant d'interface présentant des informations sur une couche transparente, permettant à l'utilisateur de les consulter sans perdre de vue l'environnement. Le concept est similaire à ce qui peut se trouver sur le cockpit d'un avion de chasse.

Localisation : La localisation est le processus qui consiste à adapter le contenu d'un jeu à une zone géographique particulière, principalement par la traduction de son contenu textuel.

MMOG : *Massively Multiplayer Online Games*, ou jeux en ligne massivement multi-joueurs. Ces jeux rassemblent un grand nombre de joueurs (des centaines, voire des milliers) sur une même instance du jeu pour une expérience coopérative ou compétitive. Ils fonctionnent généralement sur base d'un abonnement mensuel.

NPC : *Non-Player Character*, ou personnage non-joueur. Désigne un personnage d'un jeu qui n'est pas contrôlé directement un joueur humain.

PC : *Player Character*, ou personnage-joueur. Désigne un personnage d'un jeu qui est contrôlé directement par un joueur humain.

Sketch : Dessin rapide qui donne un aperçu des éléments essentiels du résultat final.

Sprite : Elément graphique, en général 2D, qui se superpose sur un environnement et peut être animé.

Chapitre 1 : Introduction

Ce rapport présente notre démarche de résolution d'un problème d'intelligence artificielle provenant du monde vidéo-ludique. Cette démarche a cherché à mettre en avant les mérites et qualités de la phase de développement dite de pré-production, qui consiste à définir, conceptualiser et modéliser l'ensemble des éléments du logiciel avant de le produire.

Ce projet est né de notre volonté de découvrir les contraintes et les problèmes que pouvait rencontrer l'intégration d'intelligence artificielle au sein d'un jeu vidéo. En effet, de par notre expérience d'utilisation de ces types de logiciel, nous avions acquis l'intuition que l'importance d'une intelligence artificielle poussée était sous-estimée par les équipes de développement.

C'est ainsi que nous nous sommes rendus « sur le terrain » au travers d'un stage en qualité de programmeur au sein d'un studio de développement de jeux vidéo, *Eversim*. Ce stage, d'une durée de 6 mois, nous a permis de nous rendre compte de la réalité du développement d'un logiciel vidéo-ludique.

Au cours de cette expérience, nous nous sommes alors rendu compte que ce n'était pas tant l'importance de l'intelligence artificielle qui était sous-estimée, mais plutôt un manque de ressources et de techniques qui constituait le problème. En effet, les problèmes d'intelligence artificielle sont souvent complexes et demandent un coût en temps et en ressources à l'exécution potentiellement élevé.

Nous avons ainsi acquis la volonté de nous documenter plus en profondeur sur le problème, au travers d'ouvrages rédigés par des gens d'expérience venant du domaine vidéo-ludique. Cette documentation a confirmé notre intuition en mettant en évidence la tension récurrente entre les phases dites de pré-production et de production – c'est-à-dire entre les phases de conceptualisation et de modélisation, et d'implémentation.

C'est alors que nous nous sommes posé la question de ce que nous pouvions réaliser, en notre qualité de technicien, afin de diminuer l'importance de ces problèmes dans le développement d'un jeu. Notre intuition fut de mettre en avant les mérites et qualités d'une phase de pré-production complète et aboutie.

Afin de mettre en pratique cette démarche, nous avons repris un problème non-résolu que nous avons rencontré au cours de notre expérience au sein du studio de développement *Eversim*. Ce problème concernait le développement d'une intelligence artificielle pour résoudre des défauts de comportements dans une simulation géopolitique.

En effet, le logiciel sur lequel nous avons travaillé, *Rulers of Nations*, consiste en une simulation géopolitique qui met en scène le monde d'aujourd'hui en proposant aux joueurs d'incarner le chef d'état d'une nation. Ils peuvent ainsi agir sur tous les aspects sociaux, économiques, politiques, environnementaux, et militaires du pays qu'ils dirigent.

Cette simulation présente cependant un défaut. En effet, les pays non-contrôlés par un joueur sont gérés par une intelligence artificielle, qui n'agit en rien pour résoudre les problèmes économiques des différents pays. Ainsi, plus la simulation avance, plus les pays non-joueurs s'enfoncent potentiellement dans une crise économique.

L'objectif de notre démarche de résolution fut donc de réaliser une intelligence artificielle qui gèrerait les aspects économiques des différents pays non-joueurs afin que ceux-ci se maintiennent à flots. Cette gestion se baserait sur l'étude des différents indicateurs à disposition de l'intelligence artificielle, comme le déficit budgétaire, et sur le choix d'actions à entreprendre, parmi toutes les actions disponibles à un chef d'état.

Cette démarche de résolution devait cependant respecter une phase de pré-production poussée, c'est-à-dire que tous les éléments des solutions proposées se devaient d'être définis, modélisés, explicités et spécifiés avant toute implémentation. Le présent document présente donc notre démarche de documentation et de résolution du problème décrit.

Au cours du premier chapitre, nous décrivons ainsi le contexte du développement d'un logiciel vidéo-ludique. Nous mettons tout d'abord en évidence les différentes compétences qui interviennent dans le développement d'un jeu vidéo, au travers des travaux de (Bethke 2003).

Nous décrivons ensuite le cycle de développement type d'un jeu vidéo, en mettant ainsi en évidence ses différences par rapport au cycle de développement d'un logiciel « classique ». Cette description se base sur les travaux de (Bates 2004).

Ceci fait, nous étudions plus en détail la question de l'intelligence artificielle au sein d'un jeu vidéo, sur base des travaux de (Funge 2004). Enfin, nous explorons les problèmes récurrents que rencontrent les développements de logiciels vidéo-ludique, grâce à l'expérience de (Bethke 2003).

Le second chapitre présente notre démarche de résolution du problème énoncé. Nous formalisons tout d'abord le problème et son environnement, avant de modéliser une solution type. Nous présentons ensuite les deux propositions de solution qui instancient cette solution type.

Le troisième chapitre décrit les choix d'implémentation que nous devons réaliser afin de porter les solutions proposées dans le logiciel cible, *Rulers of Nations*. Nous présentons donc tout d'abord l'environnement du logiciel avant de décrire l'implémentation de chaque spécification que nous avons rédigée.

Enfin, le quatrième chapitre reprend l'évaluation des solutions proposées et leur comparaison. Nous revenons ensuite sur notre démarche pour y poser un regard critique, afin de mettre en évidence ses points forts et ses points faibles.

Chapitre 2 : Contexte vidéo-ludique

Pour débuter notre analyse, nous nous intéressons tout d'abord à ce qui caractérise le développement de logiciel vidéo-ludique. Qu'est-ce qui le différencie des autres travaux de développement ? Quelles sont les compétences particulières qui entrent en jeu avant, pendant, après et autour du cycle de vie d'un jeu vidéo ? Quelle est la place de l'intelligence artificielle dans ce processus, et comment est-elle intégrée ? Enfin, quels sont les problèmes majeurs récurrents que les développements rencontrent ?

Dans un premier temps donc, ce chapitre aborde la question des métiers particuliers du jeu vidéo, en présentant un aperçu de toutes les compétences qui interviennent dans son développement. Dans un second temps, nous présenterons le cycle de développement type d'un jeu, qui possède ses phases, ses documents et ses *milestones* propres. Nous présenterons ensuite le cas particulier de l'intelligence artificielle dans le monde vidéo-ludique. Enfin, nous décrirons quelques problèmes récurrents qui peuvent entraver le processus de développement.

2.1 Les métiers[1]

Un grand nombre de compétences interviennent dans le cycle de développement d'un jeu vidéo. Au cœur de la production, nous retrouvons les concepteurs et programmeurs, ainsi que l'équipe de management, mais également les artistes tant graphiques qu'audio. Autour de la production se présentent les aspects qualité et business, sans oublier en bout de course l'assemblage, la manufacture du matériel et la production des manuels et autres guides.

Même si le travail en lui-même ne change fondamentalement pas, il n'existe pas réellement de délimitation claire quant aux différents rôles que peuvent endosser les participants au développement. En effet, chaque compagnie a sa propre manière d'agencer les compétences parmi ses équipes : chaque rôle est nommé et exécuté différemment d'une compagnie à une autre (Bates 2004).

Nous présentons donc ici non pas un relevé précis de tous les rôles qui interviennent dans le développement d'un jeu vidéo, mais un aperçu global des tâches à réaliser pour que celui-ci soit mené à bien.

2.1.1 Conception

Le métier de concepteur consiste à définir quel contenu le jeu devra posséder - quelles fonctionnalités et poussées à quelle échelle, quel scénario, quel univers, quelles règles, en somme comment le joueur vivra le jeu (aussi nommé le *game design*).

La conception comporte trois grands axes : la définition les mécaniques de jeu, la conception de missions et de niveaux, et l'écriture du scénario et des dialogues, le tout coordonné par un concepteur en chef.

[1] Cette section se base sur le chapitre 5 (*What Is a Game Made Of ?*) de (Bethke 2003).

La Figure 1 reprend le schéma des différents axes de la conception d'un jeu, que nous allons décrire.

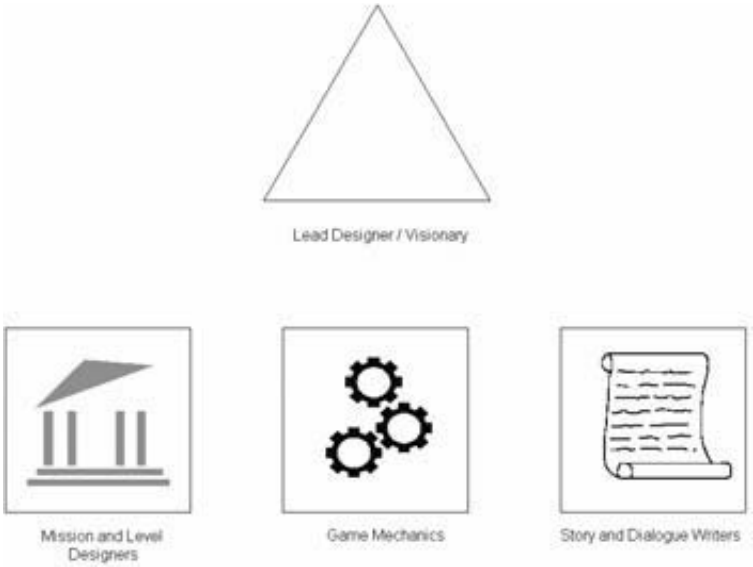

Figure 1 : Les axes de la conception d'un jeu (Bethke 2003)

Concepteur en chef

Le concepteur en chef (*Lead Designer*) joue le rôle de « chef d'orchestre » au sein de l'équipe de conception, définissant ce qui doit et ce qui ne doit pas se retrouver dans le jeu. Il détient la responsabilité de diriger l'équipe pour qu'elle produise des spécifications complètes, intéressantes et utilisables par le reste de l'équipe de développement, et ce dans un délai convenable.

Mécaniques de jeu

Un concepteur de mécaniques de jeu (*Game mechanics Designer)* a le plus souvent des compétences en programmation, car les programmeurs sont ceux qui ont le plus de chances de se charger d'implémenter ces mécaniques sous la forme de code.

Ce concepteur reçoit des directives du *Lead Designer*, interagit avec l'équipe de programmeurs pour des questions de faisabilité technique, et dialogue avec les concepteurs de niveaux et de missions afin de découvrir leurs exigences.

Enfin, un *Game mechanics Designer* jongle souvent avec des outils comme des tableurs, avec comme objectif d'équilibrer au mieux le jeu et de simuler des portions de ce jeu, ce afin de visualiser comment leurs mécaniques y interviennent.

Niveaux et missions

En général, un jeu comporte des niveaux et/ou des missions. En effet, on peut souvent décomposer un jeu en une série de défis, puzzles, niveaux ou autres missions que le joueur doit mener à bien.

Les concepteurs qui se chargent de définir ces éléments (*Levels Designer* / *Missions Designer*) peuvent être des programmeurs utilisant un langage de script pour définir ces interactions, ou des artistes construisant des cartes bloc par bloc. Il arrive également qu'ils décrivent en texte pur comment le jeu devra s'agencer, à destin d'une autre équipe.

Scénario et dialogues

L'écriture du scénario d'un jeu demande de dialoguer avec le *Lead Designer* afin de définir la direction et la découpe de ce scénario, et d'autant plus d'interactions avec les concepteurs des niveaux et missions dans le but de déterminer ce qui est possible et ce qui n'est pas faisable étant donnés les outils à disposition.

La rédaction des dialogues, quant à elle, demande au concepteur de trouver le bon ton, le bon rythme et le bon langage pour faire parler les personnages d'un monde virtuel de manière convaincante et naturelle.

2.1.2 Programmation

Le rôle des programmeurs est, de toute évidence, de produire le code qui amènera le jeu à la vie. Tout comme la conception, nous retrouvons un programmeur en chef qui peut être assisté par des directeurs techniques.

Les grands rôles incluent la programmation du cœur technique du jeu comme les mécaniques de jeu et l'intelligence artificielle, l'aspect présentation avec le moteur graphique, le support audio et l'interface utilisateur, l'aspect outils qui aideront à l'ensemble du processus, et enfin le côté réseau si nécessaire.

Programmeur en chef

Le programmeur en chef (*Lead Programmer*) est, au même titre que le *Lead Designer*, le coordinateur de l'équipe de programmation. C'est en général à lui qu'incombe les tâches de programmation les plus risquées ou les plus compliquées, en plus de son rôle de gestion.

La taille de l'équipe de programmeurs varie cependant d'une compagnie et d'un projet à un autre, ce qui donne au programmeur en chef un équilibre variable entre la production et la gestion.

Si les tâches de gestion deviennent trop importantes, ou que le *Lead Programmer* doit se charger d'une tâche importante de programmation, des directeurs techniques (*Technical Director*) peuvent aider ou prendre le relai au niveau de la coordination de l'équipe.

Mécaniques de jeu

L'implémentation des mécaniques de jeu implique un dialogue constant avec l'équipe de conception, car c'est le programmeur de ces mécaniques de jeu (*Game Mechanics Programmer*) qui va convertir la chair même du jeu en code jouable.

Son rôle sera le plus souvent de modéliser le moteur physique (ou « *physics* ») du jeu, comment les objets comme des armes ou des potions marchent, et comment les protagonistes et antagonistes fonctionnent.

Intelligence artificielle

La demande en intelligence artificielle varie entre les jeux et entre les genres (cet aspect est débattu plus en profondeur dans le chapitre 2.3 : L'intelligence artificielle). Le rôle du programmeur en charge de l'intelligence artificielle (*AI Programmer*) sera de modéliser cet aspect du jeu.

En plus de son travail sur l'intelligence artificielle, ce programmeur est en général également en charge de définir des langages de script et autres outils utilisés par l'équipe de conception.

Moteur graphique

Le programmeur en charge du moteur graphique (*3D Graphics Programmer*) s'occupe de donner vie au monde virtuel dans lequel évolue le joueur, faisant usage d'un vaste arsenal de technologies 3D pour atteindre le meilleur niveau d'immersion possible.

L'utilisation de telles techniques demande une connaissance poussée des mathématiques, principalement en calcul, algèbre, matrices, vecteurs et trigonométrie.

Support audio

Le programmeur audio (*Audio Programmer*) est en charge d'intégrer dans le jeu les éléments audio tels que les effets audio 3D, la gestion des dialogues, et la musique.

Interface utilisateur

Le programmeur en charge de l'interface utilisateur (*User Interface Programmer*) est l'expert qui gère la librairie des interfaces. Son rôle sera de faire le pont entre les différentes mécaniques du jeu au travers de l'interface utilisateur. Celle-ci comprend les contrôles, les panneaux, les menus, et les éléments du HUD[2].

[2] *Heads-up Display* ou « affichage tête haute », un composant d'interface présentant des informations sur une couche transparente, permettant à l'utilisateur de les consulter sans perdre de vue l'environnement. Le concept est similaire à ce qui peut se trouver sur le cockpit d'un avion de chasse.

Outils

L'utilisation d'outils accompagnant et aidant le développement peut faire gagner beaucoup de temps sur le processus. Qui plus est, certains outils tels que les éditeurs de contenu peuvent être délivrés en même temps que le jeu, afin de permettre aux joueurs d'ajouter leur propre contenu.

Qu'ils soient des outils aidant à la coordination des tâches, à la communication entre équipes, à l'utilisation des langages de script ou à la construction de niveaux ou de missions, le programmeur outils (*Tools Programmer*) est celui qui a la charge de les implémenter.

Réseau

Beaucoup de jeux incluent un composant réseau – que ce soit comme moyen d'obtenir des mises à jour ou de contrer le piratage, ou comme support à l'aspect multi-joueurs du jeu. Le rôle du programmeur réseau (*Network Programmer*) est d'assurer l'implémentation de ces fonctionnalités.

2.1.3 *Graphisme*

Le rôle de l'équipe de graphistes dans le développement d'un jeu est bien entendu de produire les *assets*[3] graphiques du jeu. Au sein de cette équipe, nous retrouvons le rôle de directeur graphique dont la mission est de se charger de la coordination du groupe.

L'équipe graphique comporte également le rôle de graphiste conceptuel, de graphistes 2D et 3D, et de graphiste en charge des textures, ce pour la production de modèles et éléments graphiques. Enfin, le rôle de *Storyboarder* est présent pour les besoins en cinématiques, et le rôle d'animateur pour les besoins en animations.

Directeur Graphique

Le directeur graphique (*Art Director*) coordonne l'équipe de graphistes, et s'assure que tous les *assets* graphiques sont produits en temps et en heure, tout en vérifiant que ces *assets* soient bien cohérents en termes de qualité et de thématique avec les autres *assets*.

Graphisme conceptuel

Le rôle du graphiste conceptuel (*Concept Artist*) est d'apporter un aperçu concret (le *concept art*) de ce à quoi ressembleront les *assets* graphiques une fois produits. Cet aperçu plus ou moins exhaustif permet d'obtenir la validation des producteurs et autres *stakeholders* sur la direction de la thématique d'une part, et de donner un exemple et une ligne directrice au reste de l'équipe quant à ce à quoi devra ressembler leurs productions.

[3] Ressource informationnelle pour un logiciel.

Graphisme 2D

Le rôle de graphiste 2D (*2D Artist*) comporte deux parties. La première consiste à s'occuper des arrière-plans, des portraits et autres *sprites*[4] en 2D qui composent le jeu. La deuxième consiste à produire l'aspect de l'interface utilisateur, en s'assurant que l'apparence et la navigation au sein de l'interface soit la plus agréable possible.

Graphisme 3D

De pair avec le graphisme 2D, le rôle des graphistes 3D (*3D Modelers*) est de produire les *assets* graphiques 3D du jeu. Ces *assets* peuvent être par exemple mécaniques, comme des véhicules, des vaisseaux spatiaux, ou des bâtiments ; mais ils peuvent être également organiques, comme les personnages qui animeront le jeu.

Textures

Les modèles graphiques sont généralement construits comme des assemblages de polygones sur lequel s'applique une texture qui lui donne son teint, sa couleur, son aspect. Le rôle de graphiste en charge des textures (*Texture Artist*) est de produire ces éléments.

Storyboard

Très souvent, un jeu comporte des cinématiques, des séquences animées. Le rôle d'un *Storyboarder* est de définir l'organisation de ces séquences, définissant les différentes scènes et comment elles s'articuleront, pour communiquer cet aperçu du produit aux producteurs et à l'équipe graphique au même titre que le graphisme conceptuel.

Animation

Les modèles graphiques ont bien souvent besoin d'être animés. Le rôle d'animateur (*Animator*) est présent pour remplir cette tâche. En parallèle des objets comme les antennes radar et les moulins à vent, l'animation de personnages est un domaine bien particulier et bien plus complexe.

Pour animer un personnage, on retrouve deux grandes méthodes : le *Key Framing* et la *Motion Capture*. Le *Key Framing* est une technique qui consiste à définir des *frames*[5]-clés entre lesquelles le moteur graphique introduira des transitions. La *Motion Capture* consiste à capturer les mouvements d'un être humain à l'aide d'une combinaison spéciale, et ensuite d'interpréter les données pour créer une animation similaire.

[4] Elément graphique, en général 2D, qui se superpose sur un environnement et peut être animé.
[5] Une animation, comme une cinématique, se décompose en une séquence d'images jouée à une vitesse particulière. Les images composant cette séquence se nomment des *frames*.

2.1.4 Audio

Les *assets* audio sont composés de trois grandes catégories : les dialogues, les effets sonores et la musique.

Dialogues

Les dialogues forment un des éléments les plus importants pour l'immersion ; le rôle des acteurs audio (*Voice Actors*) n'est donc pas à prendre à la légère. Un directeur des dialogues (*Voice-over Director*) peut grandement aider dans le processus de production des dialogues en choisissant les bons acteurs et le bon ingénieur, tout en gérant correctement le temps en studio.

Effets sonores

De manière comparable aux films cinématographiques, les effets sonores garnissent l'ambiance du jeu d'une touche indispensable. Le rôle d'ingénieur d'effets sonores (*Sound Effets Engineer*) est présent pour produire ces *assets*.

La production d'effets sonores peut se faire de deux manières : soit en capturant directement le son à partir de l'objet réel qui le produit, soit en partant d'un son existant et en le modifiant pour qu'il convienne au jeu.

Musique

La bande-son d'un jeu peut également être déterminante quant à l'ambiance et l'émotion qu'il créée. Certains studios font appel à des orchestres qui vont jouer la musique pour qu'elle soit capturée en direct ; d'autres font appel à des artistes audio qui produiront une bande-son synthétique à partir d'un logiciel.

2.1.5 Management

Le management d'un projet, quel qu'il soit, est évidemment toujours critique. Dans le domaine vidéo-ludique, nous retrouvons cette tâche distribuée entre plusieurs producteurs : le producteur délégué avec ses assistants producteurs, et le producteur exécutif.

Producteur délégué

Le producteur délégué (*Executive Producer*[6]) est responsable de la tâche fondamentale de la planification et de l'exécution du projet de la manière la plus profitable qu'il soit. En général, il est également responsable du studio de développement. C'est lui qui reçoit ou lance des projets de développements.

[6] Notez les faux-amis : producteur délégué (*Executive Producer*) et producteur exécutif (*Line Producer*).

Assistant producteur

L'assistant producteur (*Associate Producer*) est présent lorsque la compagnie de développement gère plusieurs grands projets à la fois. Son rôle est alors d'assister le producteur délégué dans ses devoirs de gestion pour un projet particulier, en s'occupant des tâches journalières de gestion.

Producteur exécutif

Le producteur exécutif (*Line Producer*) se charge des nombreuses tâches bénignes mais indispensables au bon fonctionnement des projets. C'est lui qui s'assurera que les documents circulent de manière rapide et efficace, que les versions du logiciel soient envoyées en temps et en heure aux testeurs et aux éditeurs, voire même que l'équipe de développement soit ravitaillée en vivres en cas d'heures supplémentaires.

2.1.6 Qualité

A l'instar des logiciels « classiques », un jeu se doit d'être testé afin de s'assurer que la qualité du produit est en accord avec les exigences. Les rôles d'assurance qualité sont donc présents pour dénicher les bugs et tout autre problème qui nuirait à la stabilité et au potentiel ludique du jeu.

L'assurance qualité d'un jeu se fait à trois niveaux principaux : par une équipe interne au studio de développement, par l'éditeur, et enfin par les joueurs.

Développeurs

La plupart des petits studios de développement n'ont pas d'équipe interne dédiée à l'assurance qualité, vu qu'elle ne serait utile que pendant une partie du projet. Il arrive donc qu'il incombe à des programmeurs de tester leur code, ou à quiconque n'a pas de tâche prioritaire à accomplir. Généralement, les assistants producteurs et les producteurs exécutifs participent également au test du produit.

Certains studios cependant emploient une équipe d'assurance qualité à plein temps, profitant du fait de mener plusieurs projets en parallèle, ce qui permet à cette équipe de ne jamais manquer de tâches à accomplir.

Editeur

Le second niveau d'assurance qualité se joue au département dédié à cette tâche au sein de l'éditeur. Ce département emploie une équipe de professionnels suivant des directives strictes du management de l'éditeur, et travaillent le plus efficacement possible pour rapporter toute erreur dans le contenu et la qualité du produit. Ces équipes d'assurance qualité sont en général menées par un directeur d'assurance qualité (*QA Lead*), qui coordonne et vérifie les rapports d'erreurs.

Le cœur de l'équipe, qui suit le jeu du début de l'assurance qualité à la maintenance après sa sortie, tourne à intervalles réguliers avec de nouveaux groupes de testeurs. En effet, après avoir travaillé sur un jeu pendant des mois, l'habitude fait qu'il est aisé de passer à côté de certains défauts fondamentaux au niveau de l'utilisabilité et du *gameplay*[7] ; changer régulièrement de testeurs contrecarre ce phénomène.

En fonction du jeu et des fonctionnalités qu'il offre, une équipe d'assurance qualité peut également comporter des branches dédiées à certains aspects tels que la partie multi-joueurs, la compatibilité matérielle avec les supports auxquels le jeu est destiné, et la localisation[8].

Joueurs

Dans les mois qui précèdent sa sortie, un jeu peu entrer dans une phase de *Beta testing*. Cette phase consiste à fournir à un nombre limité ou non de joueurs volontaires et bénévoles une version de test du produit. Ces joueurs sont en général des *fans* qui désirent avoir le premier coup d'œil sur un nouveau jeu et qui veulent saisir l'opportunité d'améliorer ce jeu avant sa sortie.

Cette phase de test est généralement supervisée par un gestionnaire désigné (le *Beta Testing Program Manager*), bien souvent un assistant producteur ou un producteur exécutif. Son rôle sera de coordonner les efforts des *Beta testers* et d'assurer la communication avec eux.

2.1.7 Business

Un jeu, bien que ludique, reste à peu d'exceptions près un produit commercial et fait partie d'une stratégie visant le profit. Dans l'univers du jeu vidéo, nous retrouvons un besoin de compétences dans le développement commercial, la promotion et la vente, et la gestion des licences commerciales si applicable.

Développement commercial

Le développement commercial (*Business Development*) est surtout de s'assurer que la direction que le studio ou l'éditeur prend est la bonne au niveau économique.

Du côté de l'éditeur, on retrouve un chargé du développement commercial (*Business Development Executive*) qui suit l'évolution de l'industrie du jeu vidéo de près. C'est lui qui donnera le premier avis sur un projet proposé par un studio, et qui décidera si le projet peut passer au comité d'acceptation (*green-light meeting*). Lors de ce *meeting*, c'est le président de l'entreprise d'édition qu'il faudra convaincre pour que le développement soit approuvé.

[7] Le *gameplay* d'un jeu reprend tous les aspects interactifs de sa conception.
[8] La localisation est le processus qui consiste à adapter le contenu d'un jeu à une zone géographique particulière, principalement par la traduction de son contenu textuel.

Du côté des studios de développement, l'aspect commercial est géré par le ou les responsables du studio (*Studio Heads*). Ces vétérans de l'industrie sont responsables de la structure et de l'environnement du studio de développement, et ce sont en général eux qui représentent les projets auprès des éditeurs.

Afin d'arranger les contrats commerciaux entre eux, les éditeurs et les studios font en général appel à des avocats spécialisés dans le domaine des affaires électroniques, ce afin de s'assurer des termes de ces contrats.

Promotion et vente

Même si cela peut sembler surprenant, dans le monde vidéo-ludique c'est l'éditeur qui absorbe tous les risques – d'un côté, le financement du développement, de l'assemblage et du marketing, et de l'autre, la vente au travers des distributeurs.

En effet, un distributeur achète une certaine quantité de copies d'un produit pour ses étagères, et s'il ne se vend pas assez vite, il peut les renvoyer à l'éditeur contre remboursement. C'est la responsabilité de l'éditeur d'évaluer en quelles quantités il proposera un jeu.

Au niveau de la promotion, il faut convaincre les distributeurs autant que les joueurs d'acheter le produit. Cela passe par les campagnes de marketing classiques, mais également par la communication avec la presse spécialisée et par les démonstrations aux *shows* incontournables tels que le fameux E3[9] aux Etats-Unis. Certains éditeurs hébergent même leurs propres conventions, afin d'être sûrs que l'accent est suffisamment mis sur leurs jeux.

Une autre force de vente pour un jeu-vidéo est le bouche-à-oreille, en particulier des *fans* les plus ardus (les *Hardcore Fans*). Ceux-ci font ressortir les meilleures qualités de leurs jeux préférés et n'hésitent pas à en faire la promotion auprès de leurs groupes de discussion en ligne et auprès de leurs amis moins *hardcores* qui leurs demandent conseil.

Licences

Il arrive qu'un jeu soit basé sur des éléments sous licences, comme des *comics*, des romans, des films, ou des stars. L'équipe de développement peut également faire usage d'outils sous licences tels qu'un moteur physique. Cet aspect peut être géré par une seule personne (le *Business Development Executive* par exemple), ou par une équipe à temps plein.

[9] L'*Electronic Entertainment Expo*, ou E3, est une convention annuelle de l'industrie du jeu-vidéo qui se tient aux Etats-Unis et qui rassemble chaque année des dizaines de milliers de visiteurs autour de centaines de stands.

2.1.8 Manuels et guides

Tout jeu est accompagné d'un manuel d'utilisation, qui décrit le contenu et les contrôles du jeu. En général, sa rédaction est confiée à un expert de l'écriture de cet aspect, qui va interagir avec l'équipe de développement dans les mois précédant la sortie du jeu afin d'apporter le contenu le plus précis possible.

Cependant, il est rare que le responsable de cet aspect ait à sa disposition toutes les fonctionnalités dans leur forme stable et finale, ce qui résulte en des descriptions parfois vagues.

Les guides de jeu (*Strategy guides*) sont là pour combler ce que le manuel n'apporte pas, c'est-à-dire les détails précis de comment le jeu fonctionne, ainsi que des tactiques et stratégies pour venir à bout du jeu.

2.1.9 Assemblage

Quand le code d'un jeu est finalisé et son manuel rédigé, il reste à produire les supports, les boîtes, à imprimer les documents, et à envoyer le tout aux distributeurs. Le rôle d'*Operations Manager* est de faire en sorte que toutes ces étapes se passent le mieux qu'il soit.

2.1.10 Manufacture du matériel

Les entreprises qui ont manufacturé le matériel qu'un jeu utilisera ont également leur mot à dire dans le développement. Principalement, ils sont là pour conseiller l'équipe de développement et s'assurer qu'elle profite du mieux possible des possibilités offertes par leur matériel.

2.1.11 Post-release

Une fois qu'un produit est délivré et vendu, il reste cependant encore des tâches à accomplir, durant cette période de *post-release*. La maintenance du produit est une chose – l'ajout ou la correction de fonctionnalités. Mais il est également nécessaire de communiquer avec les joueurs, autant en qualité de service après-vente, mais aussi sur les forums de discussion. Enfin, il faudra également combattre la piraterie et la triche, d'autant plus si le jeu comporte un aspect multi-joueurs.

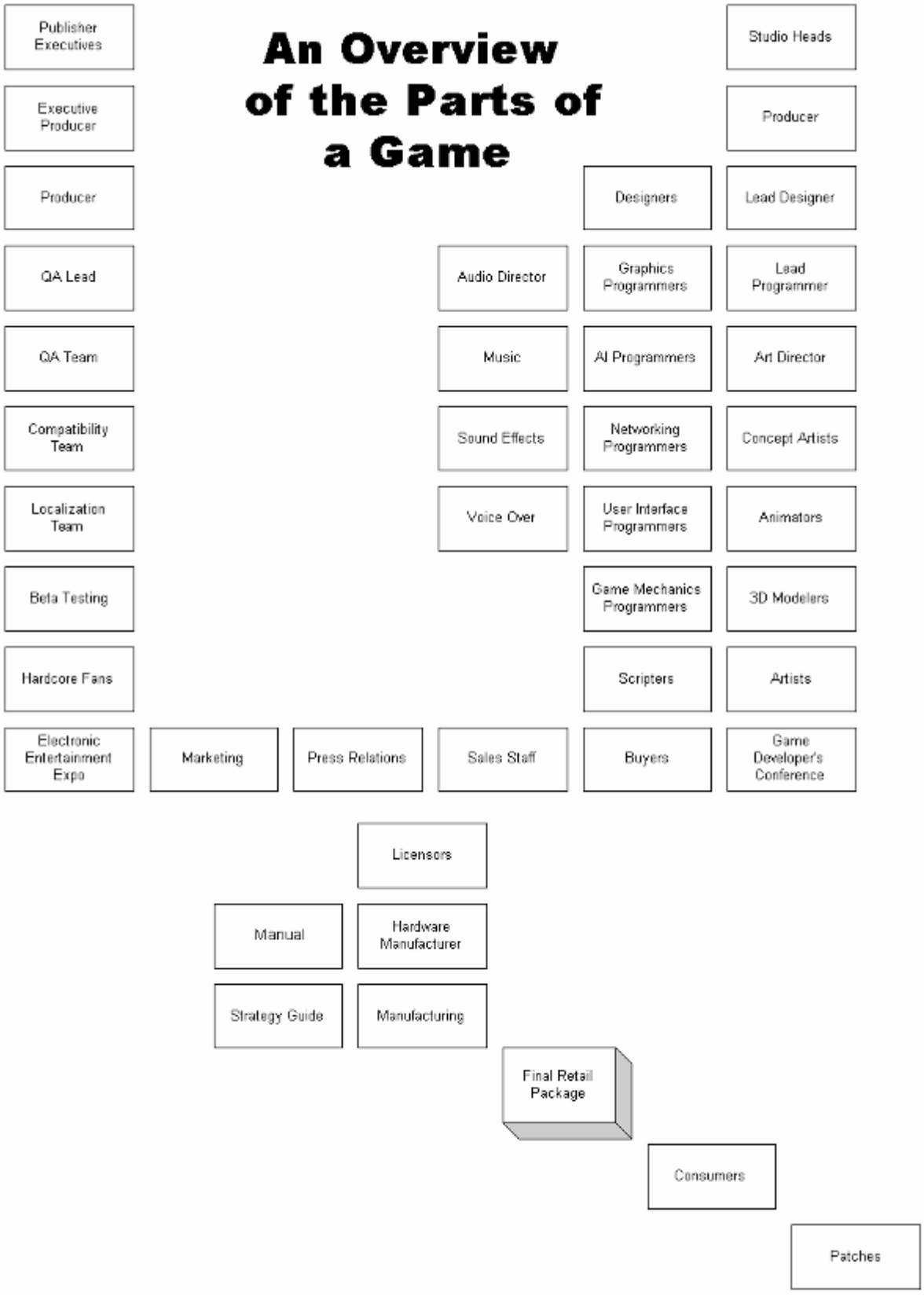

Figure 2 : Résumé des compétences intervenant dans le développement d'un jeu (Bethke 2003)

2.2 Le cycle de développement[10]

Peu importe la taille d'un projet vidéo-ludique, il passe en général par certaines phases bien précises qui sont devenues un standard dans l'industrie. Tout d'abord, un jeu se base sur une idée, un concept, qui se doit d'être développé afin de convaincre un éditeur de financer le projet.

Une fois le projet accepté par un éditeur, il s'agit de définir le planning et les ressources que le développement utilisera, et de monter un prototype. Cette étape, la pré-production, aura pour but de montrer les capacités à la fois de l'équipe et du projet.

La plus longue étape vient alors – la production en elle-même. C'est là que le jeu sera véritablement développé, codé, et testé, et traversera plusieurs *milestones*-clés jusqu'à sa sortie finale.

Enfin, une fois le produit délivré, il restera à le maintenir avec des correctifs et des mises-à-jour au cours de la période de *post-release*.

2.2.1 Concept

Cette première étape peut être considérée comme l'avant-première du *game design* d'un jeu. Durant cette période, une équipe de quelques personnes au plus cherche à développer une idée ou un concept de jeu, afin d'aboutir à quelque chose d'intéressant, d'amusant et de profitable. C'est au cœur de cette phase que les éléments-clés du *gameplay*, les premiers dessins de *concept art* et la ligne directrice du scénario seront définis.

Le développement du concept d'un jeu s'effectue au travers de trois documents-clés : le *High-Concept*, le *Pitch Doc* et le *Concept Doc*.

High-Concept

Le *High-Concept* est une description en une ou deux phrases qui décrivent les éléments-clés du jeu – typiquement, ce qui le rend attrayant et le distingue de la compétition.

Cette description est d'autant plus utile qu'elle est d'une bonne aide pour la suite du développement, en agissant comme une référence quant à quelles fonctionnalités inclure et quelles autres laisser de côté. De fait, tout ce qui ne contribue pas au cœur du concept n'est pas une piste à explorer.

Pitch Doc

Le *Pitch Doc* est un résumé de quelques pages qui décrit un peu plus en détails le concept. C'est sur base de ce document que l'équipe de développement cherchera à convaincre un éditeur de financer le projet, en montrant ce dont le jeu parle, pourquoi il marchera et comment il sera profitable.

[10] Cette section se base sur le chapitre 10 (*Project Lifecycle and Documents*) de (Bates 2004).

Concept Doc

Plus complet que le *Pitch Doc*, le *Concept Doc* étend ce dernier en un rapport de plusieurs dizaines de pages. Ce rapport sera consulté par l'éditeur après la réunion avec l'équipe de développement, afin d'avoir un aperçu plus détaillé du contenu du jeu.

Le *Concept Doc* reprend bien sûr le *High-Concept* et décrit le genre auquel appartient le jeu. Il définit également les éléments-clés du *gameplay*, en mettant l'accent sur ce qui distingue le jeu des autres du même genre. Le document comporte aussi la liste des fonctionnalités phares du jeu qui sont censées en faire un produit d'exception.

Garnie par les éléments les plus attrayants et détaillés de *concept art*, une section traite également du monde dans lequel évolue le jeu. Le scénario du jeu y trouve aussi description – les éléments-clés de l'intrigue, les héros, et leurs adversaires.

Plus terre à terre, le *Concept Doc* définit également quel cœur de marché il vise et en quoi il propose quelque chose d'intéressant pour ce marché. Dans la même veine, le document décrit quelles plates-formes le jeu prendra en compte.

Une partie essentielle du document est inévitablement l'estimation du planning et du budget requis. Cette partie décompose le développement du projet en différentes étapes et indique ce que ces étapes coûteront. Elle inclut également le relevé des pertes et profits (*Profit and Loss*, ou *P&L*), qui décrit l'estimation des coûts (salaires, équipements, licences, manufacture, marketing, promotion, distribution, …) et des revenus (les ventes du jeu et des produits associés, comme les guides). Le point clé de cette partie est la valeur du retour sur investissement (*return on investment*, ou *ROI*), qui doit montrer à l'éditeur qu'il est pertinent pour lui de prendre le risque de financer un tel projet plutôt que d'opter pour un investissement moins risqué, comme épargner.

Ensuite vient l'analyse de compétitivité, reprenant les projets en cours avec lequel le projet proposé sera en compétition, et ce qu'il a de plus qu'eux pour réussir. Si des jeux similaires sont déjà sortis par le passé et ont bien réussi, c'est le moment de montrer en quoi le projet inclut des éléments forts de ces jeux.

Le *Concept Doc* décrit aussi l'équipe qui sera chargée du développement du projet, en mettant en avant les personnes-clés et leur expérience. Le but est de donner toute confiance que l'équipe peut mener un tel projet à bien.

Enfin, le document reprend une analyse de risques, qui décrit quels sont les problèmes que le développement risque de rencontrer et comment l'équipe compte y faire face. C'est également dans cette analyse que l'équipe met en avant les éléments sûrs du développement, en particulier ceux qui comportent des risques classiques mais déjà couverts (par exemple, la possibilité de réutiliser un moteur physique déjà en place).

Le *Concept Doc* finit par un résumé qui reprend les points essentiels du jeu, et met en avant la capacité de l'équipe à délivrer un produit de qualité, à temps et dans les limites du budget.

2.2.2 Pré-production

Une fois qu'un projet a été accepté par un éditeur, il passe par une période de planification et de conception, la pré-production. Durant cette phase, l'équipe de développement va donner toute leur substance aux concepts définis durant l'étape précédente.

Les objectifs de la pré-production sont donc de compléter le *game design* dans le document du même nom, de définir le style graphique du jeu et la ligne de production des *assets* graphiques au travers de l'*Art Production Plan*, de décrire dans le *Technical Design Document* comment les fonctionnalités du jeu seront implémentées, et de décider de la planification du projet en termes de ressources, de budget et de deadlines dans le *Project Plan*, pour enfin déboucher sur un prototype du jeu qui en démontrera les éléments essentiels.

Game Design Document

Le *Game Design Document* décrit de manière exhaustive toutes les caractéristiques du jeu. Ecrit durant la pré-production et devant être constamment gardé à jour durant la production, il reprend une information la plus précise possible au sujet du *gameplay*, de l'interface, du scénario, des personnages, de l'intelligence artificielle, etc.

La clarté et la complétude de ce document sont essentielles, car il sert d'exigences de base sur lesquelles sont basés l'*Art Production Plan* et le *Technical Design Document*.

Art Production Plan

L'*Art Production Plan* décrit ce à quoi le jeu ressemblera et comment les graphismes qui le composent seront développés.

Le style graphique et la direction artistique sont définis par le *Lead Designer*, l'*Art Director* et le *Concept Artist* du projet. Ce dernier réalise alors des feuilles de références à partir desquelles les autres artistes peuvent travailler, ce qui permet de s'assurer de la cohérence du style graphique. Ces références constituent la bible graphique (*Art Bible*) du projet.

L'*Art Production Plan* reprend également la ligne de production des *assets* graphiques – c'est-à-dire le processus de conversion d'un concept en un élément graphique concret du jeu. Par exemple, créer un personnage passe par sa description par le concepteur, puis par un *sketch*[11] par le *Concept Artist*, puis par un modèle 3D, qui sera ensuite texturé, puis animé, avant de se voir attribuer une IA et d'être invoqué dans le jeu. Tous les outils utilisés dans ce processus doivent être compatibles entre eux – pouvoir communiquer de manière à ce que chaque *output* d'une étape puisse être utilisé comme *input* de la suivante.

Enfin, l'*Art Production Plan* donne une première version de l'estimation des coûts de l'équipe graphique, de ses tâches et de l'équipement qui lui sera nécessaire.

[11] Dessin rapide qui donne un aperçu des éléments essentiels du résultat final.

Technical Design Document

Le *Technical Design Document* reprend la ligne directrice suivant laquelle l'équipe technique transformera le texte du document de conception en code exécutable. Il établit l'aspect technique de la ligne de production de l'*Art Production*, définit les tâches de toutes les personnes impliquées dans le développement, et estime combien de temps prendront ces tâches.

Ce document reprend de même les principaux outils qui seront utilisés pour construire le jeu, détaillant ceux que le studio possède déjà et ceux qui devront être achetés ou créés. Il décrit également le matériel et les logiciels qui devront être acquis, et tout autre changement de l'infrastructure du studio nécessaire pour les besoins du développement.

Projet Plan

Le *Project Plan* est la feuille de route qui décrit comment le jeu sera développé. Il reprend la liste des tâches définie dans le *Technical Design Document* et la transforme en un planning véritable.

Ce document décrit donc l'estimation du personnel qui sera impliqué dans le développement, quand leur travail débute, et à combien s'élèvera leur salaire. Il reprend aussi l'estimation des ressources externes nécessaires pour le projet – matériel, voix, musiques, vidéos, etc. Ces estimations conduisent au calcul du budget requis par le projet, ce qui permet de mettre à jour l'aspect pertes et profits (*P&L*) important pour l'éditeur.

Le *Project Plan* n'en serait pas un sans la définition d'un planning pour le développement. Ce planning est décomposé en *milestones* qui se doivent d'être précises et binaires – la réalisation d'une *milestone*, un délivrable, est soit complet, soit incomplet. Cela permet d'estimer une planification du projet qui sera très utile aux équipes de promotion et de marketing pour prévoir leurs campagnes.

Enfin, de pair avec le planning, on retrouve la description de la méthode qui permettra de se rendre compte de l'avance ou du retard du projet. Généralement, un diagramme de Gant révélant le chemin critique et les dépendances entre les tâches est utilisé à cet effet.

Prototype

Le délivrable le plus tangible de la pré-production est le prototype. Le prototype est un fragment du logiciel final qui en capture les éléments essentiels, ceux qui font que le jeu est amusant et intéressant, et le mettent en avant par rapport aux autres jeux.

Le prototype est souvent utilisé par les éditeurs pour éprouver le concept et l'équipe de développement, et est donc un élément déterminant dans le processus. Il se doit de montrer les aspects les plus importants du concept, comment les plus grandes inconnues et les plus grands risques sont gérés par l'équipe, et à quel point l'équipe est apte à délivrer du contenu de qualité.

2.2.3 Production

La production est la partie la plus longue et la plus ténue du processus de développement. C'est durant cette phase que tous les plans de productions sont mis en œuvre, que les *assets* sont créés ou acquis, et que le code est produit.

D'une durée variable – généralement entre 6 mois et 2 ans – cette étape comporte plusieurs grandes *milestones*. La première est l'*Alpha* – ou la première version jouable du produit, et ce du début à la fin. La seconde est la *Beta*, où tous les *assets* sont fixés et le développement arrêté, et où la seule préoccupation est la correction de bugs. Durant les derniers jours de la *Beta*, on entre dans la période de gel du code (*Code Freeze*) où seuls les défauts majeurs sont corrigés, avant que le jeu ne soit déclaré délivrable (*RTM* ou *Release to Manufacture*).

Alpha

La définition d'une version *Alpha* d'un jeu varie d'une compagnie à une autre, mais généralement elle désigne la première version du jeu jouable du début à la fin. Il peut toujours exister quelques défauts ou éléments manquants, et tous les *assets* ne sont sans doute pas finalisés, mais le moteur, l'interface et tous les autres sous-systèmes majeurs sont complets.

L'*Alpha* marque le début d'une période où le *focus* n'est plus tant sur la construction que sur la finalisation, où l'on décide si on doit abandonner certaines fonctionnalités pour délivrer à temps, et où les premiers testeurs externes à l'équipe technique éprouvent le jeu à la recherche de bugs.

Beta

Quand un jeu atteint la version *Beta*, tous les *assets* sont intégrés, tout développement s'arrête, et les seules tâches restantes concernent la correction de bugs. Quelques éléments mineurs peuvent encore évoluer, mais l'objectif est à présent de stabiliser le projet pour sa sortie et d'en éliminer le maximum de défauts. C'est également le moment de tester le logiciel sur les plates-formes sur lesquels il est censé tourner, afin de détecter toute incompatibilité.

La dernière portion de la *Beta* est appelée *Crunch Time*. Au cours de ces semaines, l'équipe de développement preste un grand nombre d'heures supplémentaires afin de s'assurer au possible que le produit est stable, complet et prêt à être délivré. C'est en général une période de grande pression inévitable pour tout projet, et sa gestion en est d'autant plus critique.

Code Freeze

Durant les derniers jours de la *Beta*, le code est gelé – seuls les défauts majeurs découverts dans les derniers tests seront corrigés. La grande majorité du travail est terminé, et la préparation des versions candidates à la distribution commence.

RTM

Lorsqu'une version candidate à la distribution a été testée exhaustivement et a été déclarée comme acceptable, le jeu est déclaré prêt à être assemblé (*Release to Manufacture*, ou *RTM* ; on dit également que le jeu en est à sa version *Gold*).

2.2.4 Post-release

Une fois la production enfin terminée, le produit entre dans une phase de maintenance, dont la durée et la profondeur varient entre les produits et les studios. Au cours de cette phase, des correctifs et des améliorations peuvent être déployés.

Correctifs

Même une version *Gold* peut toujours contenir des défauts qui n'ont pas été découverts durant la phase de test, ou qui ont été tout simplement ignorés car peu prioritaires. Des problèmes de compatibilité matérielle peuvent également être découverts, surtout dans l'univers du PC – il est en effet impossible de tester toutes les combinaisons de BIOS, carte graphique, carte son, moniteur, processeur, système d'exploitation, clavier, souris et joystick possibles. Les développeurs sortent dont en général des correctifs (*Patches*) après la sortie du jeu pour remédier à ces problèmes.

Améliorations

Un jeu peut également recevoir des améliorations même après sa sortie. Ces améliorations ajoutent du contenu au jeu afin de l'améliorer ou d'allonger sa durée de vie. C'est une stratégie indispensable sur certains produits qui se doivent de tourner longtemps, comme les MMOG[12].

[12] *Massively Multiplayer Online Games*, ou jeux en ligne massivement multi-joueurs. Ces jeux rassemblent un grand nombre de joueurs (des centaines, voire des milliers) sur une même instance du jeu pour une expérience coopérative ou compétitive. Ils fonctionnent généralement sur base d'un abonnement mensuel.

2.3 L'intelligence artificielle[13]

Le jeu vidéo est un domaine particulier de l'intelligence artificielle, principalement car les contraintes et les buts sont eux-mêmes particuliers. Ce chapitre donne un aperçu de ce qui rend ce domaine particulier.

Tout d'abord, nous parlerons des personnages faisant partie d'un jeu, car ce sont eux qui sont le siège de l'IA dans ce domaine. Nous décrirons ensuite la notion de comportement, concept-clé du domaine. Nous montrerons ensuite comment l'IA d'un jeu vidéo interagit avec son environnement à l'aide d'une architecture type. Enfin, nous mettrons en évidence les différences entre l'IA du domaine académique et l'IA du domaine vidéo-ludique.

2.3.1 Personnages

Il est intéressant de faire la différence entre un personnage joueur (*player character* ou *PC*) et un personnage non-joueur (*non-player character* ou *NPC*). Un personnage joueur est un personnage dont le comportement est contrôlé par un humain à l'aide d'un périphérique d'entrée comme un *joystick*. Un personnage non-joueur est, par opposition, un personnage contrôlé par une intelligence artificielle et qui agit donc de manière autonome. Les *PC* sont bien souvent les héros du scénario, et les *NPC* leurs alliés et leurs adversaires.

La distinction entre un *PC* et un *NPC* n'est cependant pas toujours évidente. Ainsi, certains jeux permettent au joueur de contrôler plusieurs personnages à la fois ou à des moments différents. Dans ce cas, les personnages qualifiés de *PC* et ceux qualifiés de *NPC* changent constamment. Par exemple, dans certains jeux le joueur contrôle une équipe de personnages, et passe de l'un à l'autre de manière si fluide que la distinction en devient compliquée. Ces personnages agissent donc come des *NPC* jusqu'à ce que le joueur leur donne un ordre, qu'ils exécuteront avant de retourner à leur comportement autonome.

Un autre élément moins visible mais tout aussi présent est qu'un *NPC* peut également endosser le rôle de *cameraman*, de technicien du son ou des lumières, voire même de réalisateur. Il est bien sûr rare qu'un personnage tenant une caméra ou qu'un réalisateur soit visible à l'écran, mais cela n'empêche qu'un bout de code est derrière ces rôles et les contrôle. Il est pertinent de considérer ces rôles comme des *NPC*, vu qu'ils partagent les mêmes propriétés essentielles.

2.3.2 Comportement

Tout personnage dans un jeu est associé à au moins un contrôleur, et des contrôleurs peuvent être partagés entre différents personnages. Un contrôleur agit comme s'il était le cerveau du personnage : en entrée, il reçoit des informations sur l'état du monde virtuel, et en sortie, il produit des choix d'actions qui affecteront le monde et produiront le comportement associé.

[13] Cette section se base sur le chapitre 1 (*Introduction*) de (Funge 2004).

Pour des personnages joueurs, le contrôleur consiste en l'interprétation des signaux envoyés par la manette de jeu. Techniquement, le cerveau du joueur humain fait également partie du contrôleur, vu que c'est de là que viennent les décisions d'actions.

Pour les personnages non-joueurs, le contrôleur peut prendre des formes diverses et variées, en fonction de la technique utilisée et du résultat attendu. Mais ce qui compte avant tout, c'est le comportement que le contrôleur simulera, vu que ce n'est que cette partie du programme que le joueur peut visualiser. Plusieurs approches d'implémentation d'un contrôleur peuvent donc déboucher sur le même comportement, et donc être relativement équivalentes aux yeux du joueur.

Cela est heureux, car l'implémentation de tels contrôleurs comporte un grand nombre de limitations techniques – il n'est pas question de prendre plusieurs secondes à prendre une décision quand une dizaine d'autres contrôleurs font la même chose et ce au cœur d'un jeu d'action rapide. Cette équivalence par le comportement donne une grande flexibilité aux développeurs quant au choix de l'implémentation d'un contrôleur.

Par exemple, pour se déplacer d'un point à un autre dans l'environnement virtuel, un contrôleur peut faire appel à un algorithme de recherche qui calculera à la volée le chemin qu'il devra suivre. Une autre solution est de placer à des endroits prédéfinis des « panneaux de signalisation ». Ces panneaux, invisibles pour le joueur, indiquent aux *NPC* par où ils doivent passer selon leur intention de destination.

Ces solutions débouchent sur une perception de comportement relativement équivalente par le joueur, mais elles comportent chacune leurs avantages et leurs défauts : la planification à la volée est évolutive et élégante, mais coûte cher en ressources *CPU* ; tandis que l'utilisation de panneaux est simple et demande peu de compétences à utiliser, mais est laborieuse et n'est pas évolutive.

Le choix de l'implémentation d'un contrôleur doit donc se faire au cas par cas, selon les exigences du produit, les capacités matérielles et logicielles, et le degré d'expertise des développeurs, voire même selon leurs préférences personnelles.

Rationalité et entropie

Ce qui intéresse le joueur dans sa perception des *NPC* est le degré de rationalité de leur comportement. En effet, un contrôleur qui crée des comportements « idiots » peut nuire à l'immersion et à la qualité globale du produit.

Il convient cependant de s'intéresser à ce qui rend un comportement rationnel. De fait, les objectifs d'un *NPC* peuvent être irrationnels en eux-mêmes – un *NPC* peut très bien sortir de sa cachette et attaquer le personnage joueur qui pourtant est apparemment bien plus fort et qui vient de battre tous les coéquipiers du *NPC*. Un tel comportement peut sembler irrationnel, mais peut cependant être en accord avec le rôle du *NPC*.

L'objectif d'un *NPC*, au final, est de contribuer à l'expérience ludique du jeu. A cet effet, le contrôleur peut être programmé pour donner des actions telles que tenter d'arrêter le joueur à tout prix. Ces actions sont définies par les concepteurs qui espèrent alors produire une expérience ludique intéressante.

Ce qui est considéré comme rationnel dépend donc du jeu, du niveau de difficulté, du contexte dans lequel le joueur évolue, et même des attentes du joueurs. Il est loin d'être évident d'identifier un comportement rationnel et encore moins de le programmer, mais il est cependant aisé de reconnaître la plupart des comportements irrationnels – par exemple, deux *NPC* se bousculant continuellement en essayant d'atteindre une échelle en même temps. C'est précisément ce genre de comportement qu'il faut éviter de simuler.

Un autre élément important du comportement d'un *NPC* est la notion d'entropie. Afin de fournir une expérience de jeu sans cesse intéressante et changeante, il convient de faire en sorte que le comportement des *NPC* ne soit pas toujours le même, qu'importe le nombre de fois que l'on rejoue une partie du jeu. Il est donc intéressant d'ajouter une part d'aléatoire, d'imprévisibilité dans la création des comportements des *NPC*. Cette imprévisibilité est l'entropie d'une IA, qu'il convient d'équilibrer correctement afin que les comportements ne soient ni si aléatoires qu'aucune réelle intelligence ne peut être garantie, ni si rigides que le joueur peut prévoir la réaction d'un *NPC* à tous les coups.

2.3.3 Architecture

La Figure 3 reprend une architecture type pour un jeu vidéo, décrivant les composants principaux et leurs interactions. Tous les jeux ne répondent pas précisément à une telle architecture, mais ils conservent des composants relativement similaires.

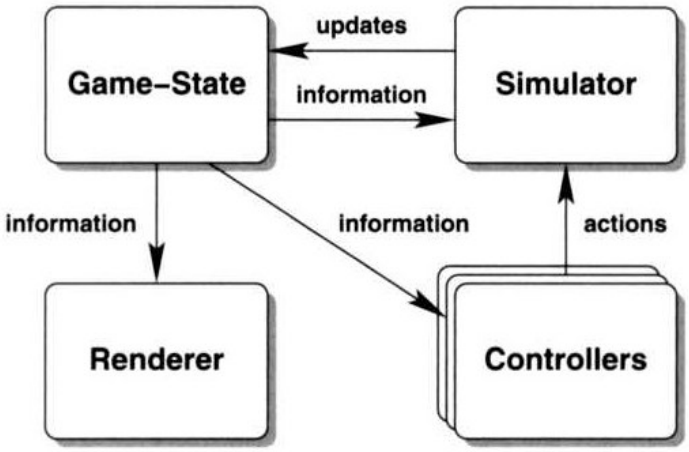

Figure 3 : Architecture type d'un jeu vidéo (Funge 2004)

L'état du jeu (*Game-State*) représente l'état courant du monde virtuel du jeu. Il reprend toutes les informations sur les objets le composant et donne accès à ces informations de sorte qu'elles puissent être interrogées par les autres composants à propos de leur état.

Le simulateur (*Simulator*) contient les règles qui définissent comment l'état du jeu change – en d'autres termes, les règles physiques du jeu. Associé à un ensemble d'animations, c'est donc le simulateur qui est responsable de l'animation d'un personnage en réaction à l'action choisie par son contrôleur.

Le gestionnaire des rendus (*Renderer*) est responsable du rendu d'une représentation de l'état du jeu, dont la forme consiste généralement en des images et des sons.

Les contrôleurs (*Controllers*) régissent chacun un ou plusieurs personnages du jeu, et sont responsables du choix des actions de ces personnages. Pour les *PC*, le contrôleur se fait l'interprète des boutons sur lesquels le joueur appuie. Pour les *NPC*, le contrôleur est l'intelligence artificielle qui compose le « cerveau » du personnage.

Perception

La capacité d'un personnage de percevoir le monde qui l'entoure est aussi fondamentale que sa capacité à agir au sein de ce monde. Sans cette perception, le contrôleur ne peut pas décider quelle action est appropriée, ou même quels effets une action aura.

Pour le *PC*, le *Renderer* fournit les informations nécessaires au joueur pour qu'il perçoive ce qui se passe dans le monde du jeu. Les *NPC*, eux, interagissent avec l'état du jeu afin d'obtenir les informations qui leur sont nécessaires. Afin de faciliter l'écriture d'un contrôleur de *NPC* et afin que la perception de son environnement par le contrôleur soit cohérent avec les attentes du joueur, il est nécessaire de trouver le bon niveau d'abstraction pour les informations reçues du *Game-State*.

Décisions

Lorsqu'un *NPC* est capable de percevoir le monde qui l'entoure, il doit pouvoir être capable de prendre des décisions, qui dépendent des informations qu'il reçoit. Une méthode d'implémentation d'un contrôleur est d'en faire un contrôleur réactif, c'est-à-dire qui réagit directement à l'information que lui donne le *Game-State* en prenant des décisions sur le fait.

Il est cependant nécessaire d'ajouter une mémoire à de tels contrôleurs – à défaut, ils risquent de tourner en rond, tant figurativement que littéralement. Il est donc pertinent pour un contrôleur de maintenir un état interne qui se souviendra de décisions passées, de sorte à ce qu'il prenne des décisions non pas seulement par rapport à ce qui se passe, mais aussi par rapport à ce qu'il s'est déjà passé.

Un contrôleur efficace peut très bien prendre ses décisions suivant un ensemble de règles prédéfinies. Une autre solution consiste à lui donner un but et un ensemble d'actions possibles, et à le laisser chercher comment il atteindra ce but à l'aide des actions qui lui sont données. Pour ce faire, un contrôleur a besoin d'avoir accès à un modèle du monde virtuel qu'il peut utiliser afin de prédire l'effet des actions avant qu'il ne les exécute.

Un tel modèle peut être fourni de deux manières : soit en donnant accès au *Simulator* par le contrôleur, soit en lui donnant une représentation approximative du monde. Toutes ces solutions diffèrent en termes de temps et de précision, et leur utilisation dépend donc du cas que l'on veut gérer.

Il est également envisageable pour un contrôleur d'apprendre des événements passés dont il a souvenance grâce à son état interne. L'idée d'apprentissage consiste à donner à un contrôleur la capacité de généraliser des cas précédemment rencontrés à des nouveaux cas qu'il n'a jamais rencontré.

2.3.4 Game AI

On utilise parfois la dénomination *Game AI* pour distinguer l'intelligence artificielle utilisée dans les jeux vidéo de l'intelligence artificielle du milieu académique. Une grande différence est que l'intelligence artificielle dans un jeu vidéo ne se doit pas d'être générique. Son but n'est pas non plus nécessairement de faire avancer la compréhension de l'être humain au travers d'un travail technique.

Bien souvent, une solution de *Game AI* est acceptable pour autant que le comportement qu'elle génère est jugé plausible dans le faible nombre de cas auxquels elle est appliquée. Il existe néanmoins des avantages économiques clairs quant à la génération de solutions les plus génériques qu'il soit – ne fut-ce que pour la réutilisabilité des *assets*.

Une autre différence entre le milieu académique et vidéo-ludique est la définition de ce que l'on considère comme de l'intelligence artificielle. De fait, il existe un vieil adage en intelligence artificielle académique qui dit qu'une fois qu'un problème d'IA a été résolu, il n'est plus à considérer comme un problème d'IA.

En *Game AI*, on parle généralement d'intelligence artificielle quand on discute de problèmes de contrôles. Par exemple, déterminer les angles de mouvement de sorte que la main d'un personnage se déplace vers un point donné est parfois considéré comme un problème d'IA, alors qu'en-dehors des jeux, ce ne serait qu'un simple problème de robotique, déjà fort bien étudié et compris.

Il est cependant parfois pertinent, voire requis, de développer des intelligences qui pourront générer des comportements haut-niveau, d'une manière plus en phase avec les problèmes académiques.

Une approche typique de *Game AI* pour créer de telles intelligences est de construire un contrôleur possédant une grande quantité de connaissances à propos des comportements qu'il devra générer. Il est alors similaire à un système expert, dont le domaine d'expertise serait le choix des actions qu'un *NPC* va réaliser à un moment donné.

Cependant, une telle approche souffre de tous les défauts d'un système expert classique. De fait, devoir imaginer toutes les situations face auxquelles le contrôleur devra réagir implique qu'on en oubliera certainement quelques-unes, ce qui peut déboucher sur un comportement irrationnel dans le cas d'une situation inconnue du contrôleur.

Le second problème avec ce type d'approche est l'évolutivité – si un comportement irrationnel est détecté, ou si le monde évolue, il faut rajouter de la connaissance au système. Le monde virtuel pouvant se révéler fort complexe, cette tâche peut prendre un temps non-négligeable.

Cependant, de tels systèmes experts peuvent très bien fonctionner sur des cas réduits et précis. Or, un jeu est bien souvent en lui-même un cas réduit et précis – ce qui fait que cette approche a déjà porté ses fruits dans le domaine et peut toujours se révéler pertinente.

Néanmoins, les jeux se révèlent de plus en plus complexes, et les exigences en termes de qualité augmentent également – cela rend l'approche système expert de moins en moins envisageable. Pour résoudre ce problème, on se tourne de plus en plus vers des techniques de recherche dynamique ou d'apprentissage, certes moins simples mais bien plus évolutives.

Un dernier rapprochement avec le monde académique est la notion d'incertitude. Dans un monde virtuel simulé, il est rare que des capteurs comprennent du bruit, mais d'autres sources d'incertitudes existent. De fait, il existe une incertitude inhérente quant aux états futurs du monde virtuel – soit par les actions du joueur, soit parce-que la simulation devient trop complexe pour être prévisible à long terme – et de l'incertitude peut même être ajoutée artificiellement afin d'accroître le réalisme.

Pour modéliser et résoudre ces problèmes d'incertitudes[14], certaines techniques du monde réel peuvent être appliquées aux mondes virtuels – par exemple, les représentations sous forme de graphe. Ces représentations sont également au cœur de certains algorithmes d'apprentissage, ce qui peut leur donner une double utilité.

Game Design

La technologie sous-jacente n'est qu'un des facteurs qui influencent la qualité de l'IA dans un jeu. Dans certains produits, le développement de l'IA est tout simplement plus compliqué, du fait de ses objectifs.

Cependant, la conception de tels jeux peut être réalisée ou modifiée avec le travail sur le composant IA en tête, de sorte à rendre la tâche plus aisée. Par exemple, restreindre l'espace de la caméra du jeu résulte en un nombre plus réduit de *NPC* visibles à l'écran, ce qui simplifie la gestion des contrôleurs.

Un autre point est qu'au sein d'un jeu, un contrôleur même régi par des règles simples peut donner lieu à une intelligence imprévue mais intéressante. Ce phénomène, appelé l'émergence, est une arme à double tranchant – cela enrichit l'entropie, mais au prix d'un contrôle moins précis sur les comportements générés.

Les animations peuvent également aider à rendre une IA plausible aux yeux d'un joueur. Par exemple, imaginons un *NPC* encerclé par des monstres et dont le contrôleur ne parvient pas à trouver quoi que ce soit d'intelligent à faire. Au lieu que le *NPC* reste sur place sans faire quoi que ce soit, le système d'animation pourrait prendre le relai et jouer une animation montrant le *NPC* crier, apeuré. Même si le contrôleur n'a rien trouvé de pertinent à faire, le joueur n'a cependant décelé aucun comportement irrationnel.

Le problème de la *Game AI* est donc à étudier au cas par cas, en fonction du jeu, de sa conception, et des attentes de l'équipe.

[14] Attention cependant à ne pas résoudre l'incertitude ajoutée artificiellement, au risque de définir un modèle inutile.

2.4 Les problèmes récurrents[15]

L'industrie du jeu vidéo se frotte bien souvent à une série de problèmes récurrents, qui ne sont pas sans rappeler les problèmes similaires dans l'industrie du logiciel « classique ». En effet, comme montré par le rapport « Chaos » annuel du *Standish Group*[16], une majorité des projets informatiques dépasse leurs délais et leur budget, avec un bon tiers qui se retrouvent tout simplement annulés en cours de production.

Tout comme l'ingénierie de logiciels classique, des standards et des bonnes pratiques commencent à se développer et à donner de la rigueur au développement vidéo-ludique. D'un côté comme de l'autre, ce ne sont en effet pas les méthodes qui manquent, mais plutôt le crédit qu'on leur accorde et la volonté de les appliquer.

Mais bien que l'industrie vidéo-ludique rencontre des problèmes similaires tels que les budgets et les délais, elle comporte des caractéristiques inhérentes qui donnent lieu à des problèmes tout à fait nouveaux, tels que la tension entre la pré-production et la production.

Ce chapitre reprend donc les problèmes-clés qui nous ont paru les plus essentiels quant à notre analyse du domaine vidéo-ludique, et pose la question de ce que nous, en tant que technicien, pouvons réaliser afin de réduire l'importance de ces problèmes.

2.4.1 Les moyens

Un des problèmes majeurs de l'industrie du jeu vidéo est que très peu de titres finissent par générer du profit. Sur les milliers de jeux qui garniront les étagères des distributeurs, seuls quelques centaines génèreront du profit, et quelques dizaines seulement rapporteront un montant significatif aux éditeurs et développeurs.

Bien que ce fait puisse être quelque peu effrayant, il témoigne de l'échec de nombreux projets, et de l'incapacité de certaines équipes de développement à estimer le budget nécessaire pour mener leur projet à bien.

De fait, il est nécessaire que l'équipe ait pleine conscience de ce qu'elle attend du produit en termes financiers – s'attend-elle à faire du profit ? Compte-t-elle sur les *royalties* pour se tenir à flots ? Est-ce un éditeur ou un investisseur indépendant qui financera le développement ?

Ces questions se révèlent critiques pour que le projet réussisse. C'est sur base des réponses à ces questions que les estimations de budgets de personnel, de ressources et de durée pourront être formulées. Sauf dans le cas où le projet a des fonds quasi-illimités, il est plus que nécessaire d'apporter une estimation la plus réaliste qu'il soit quant à ce que le projet demandera pour être réalisé – mieux vaut se rendre compte qu'on vise trop gros ou qu'on demande trop peu avant de se lancer dans la production, plutôt qu'en fin de cycle.

[15] Cette section se base sur le chapitre 3 (*What Makes Game Development Hard?*) de (Bethke 2003).
[16] http://www.standishgroup.com/

Un des grands défauts dans la conception de jeux est de fait le *feature storm* – ou l'excitation autour d'un concept qui pousse à ajouter fonctionnalité sur fonctionnalité, pour au final se retrouver avec un mastodonte impossible à financer ni même à réaliser.

La Figure 4 résume le problème des moyens dans l'industrie vidéo-ludique.

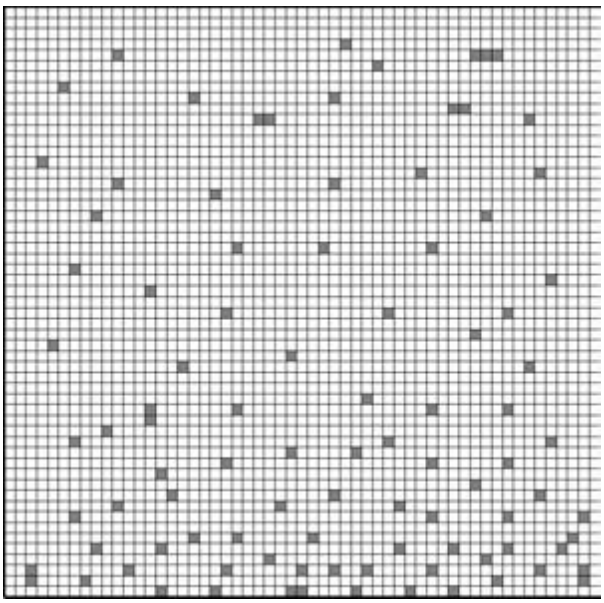

Figure 4 : Les rares succès parmi les sorties annuelles de jeux (Bethke 2003). Chaque carré représente une sortie, l'ensemble des carrés illustrant les sorties sur une année. Un carré blanc représente un titre qui n'a donné aucun profit, un carré gris représente un titre qui a rapporté de l'argent.

2.4.2 Les délais

A de rares exceptions près, une équipe de développement est constamment sous pression pour sortir un jeu dans les délais les plus courts qu'ils soient. Il ne s'agit pas seulement de respecter le planning : l'industrie vidéo-ludique est un milieu très compétitif, et si un développeur prend trop de temps à sortir un concept, un autre peut très bien le « coiffer au poteau » - sans parler de l'évolution technologique, principalement le côté graphique, qui aiguise sans cesse les attentes des joueurs en terme de qualité.

Il est en effet rare qu'un projet qui dépasse de loin son budget et son planning rencontre du succès. Le Tableau 1 reprend quelques titres, leur durée de développement et leurs résultats.

Titre	Durée	Résultats
Stonekeep 1	5 ans	Faibles ventes
Daikatana	4 ans + forts surcoûts	Faibles ventes
Messiah	5 ans	Faibles ventes
Max Payne	5 ans	Fortes ventes
The Sims	5 ans	Ventes exceptionnelles
Baldur's Gate	Plus de 3 ans	Fortes ventes
Duke Nukem Forever[17]	Plus de 10 ans	Toujours en développement
Stonekeep 2	5 ans	Projet annulé
Ultima Online 2	4 ans	Projet annulé

Tableau 1 : Projets dont la durée planifiée a été largement dépassée

Les titres qui n'ont pu percer ont échoué face à la compétition de jeux plus intéressants sortis plus vite et pour un coût moins important. Les rares jeux qui ont réussi ont eu la chance de profiter d'un concept innovant que personne n'avait pu encore développer à ce jour – le meilleur exemple étant *Les Sims*, un jeu dont le cœur de cible est le public féminin, que très peu d'éditeurs considéraient alors.

Il est donc plus qu'important d'estimer et de planifier correctement un développement. Décomposer le processus en une série d'étapes mesurables, claires, précises et bien définies, accompagnées de leur estimation en coût, en ressources et en temps, et garder ce planning à jour et constamment vérifier que le projet est dans les temps sont d'autant de stratégies critiques que beaucoup de développeurs bâclent.

2.4.3 La pré-production

Les ponts supportent généralement bien leur charge tout au long de leur durée de vie. Les barrages se brisent rarement en inondant des villes entières. Pourquoi les projets d'ingénierie civile semblent régulièrement réussir alors que les projets d'ingénierie logicielle dépassent régulièrement leur budget, leurs délais, et leur qualité attendue ?

La différence provient des processus et des méthodes. Réaliser quelque chose de complexe faisant intervenir nombre d'humains compétents durant une longue période de temps demande de décomposer la tâche complexe et imposante en une série de petites tâches réalisables et mesurables.

Idéalement donc, définir ce que l'on va réaliser doit venir avant sa réalisation effective – et dans l'univers du jeu vidéo, cette étape est la pré-production. Cette phase est malheureusement bien souvent bâclée par les développeurs.

[17] A noter qu'au moment de la publication de ce document (mai 2011), *Duke Nukem Forever* a atteint sa version *Gold* et sa sortie est prévue en juin. Le développement du jeu est passé par plusieurs studios et a été recommencé plusieurs fois également. Il est à ce jour le jeu qui aura eu la durée de développement la plus longue de l'histoire du jeu vidéo : 14 ans.

En effet, pour bien faire, la pré-production devrait comporter la distillation de toutes les exigences du jeu, l'analyse des conséquences de ces exigences, une phase de tamisage pour répondre aux attentes commerciales du studio, et une conception détaillée des éléments de *gameplay*, graphiques, audio et techniques.

Cette phase devrait être itérative jusqu'à ce que l'ensemble des exigences ait été relevé. Le document de pré-production devrait également reprendre une analyse complète des risques et comment l'équipe de développement compte les gérer. Enfin, ce document devrait être partagé avec toutes les parties impliquées – développeurs, éditeur, équipes de marketing, de promotion et de vente.

Si la période de pré-production est menée de manière exhaustive, précise et complète, la période de production devrait se dérouler sans surprise – avec le minimum de *crunch time*. Malheureusement, la période de pré-production est souvent sous-estimée par les développeurs.

En effet, pressés par la compétition, l'industrie tend à compresser le plus possible la période de pré-production. Cette période n'est donnée aucune visibilité, honneur ou célébration spéciale dans l'industrie – elle est une période certes indispensable, mais dont on doit être débarrassée au plus vite afin de produire quelque chose de concret.

Heureusement, l'industrie mûrit et les mentalités changent progressivement. De plus en plus d'éditeurs revoient leur manière d'accepter des projets, remplaçant ces courtes réunions d'évaluation d'un projet par des analyses plus poussées de son contenu et du processus que compte mettre en place l'équipe de développement.

2.4.4 Quelle réponse ?

Les problèmes de moyens, de délais et de pré-production représentent la plupart des problèmes majeurs que rencontre l'industrie du jeu vidéo. En notre qualité de technicien, nous nous posons donc la question de ce que nous pouvons réaliser ou changer afin de réduire l'importance de ces problèmes.

L'avis que nous choisissons d'émettre est qu'une réflexion et un recul sont nécessaires avant de se lancer dans toute implémentation. Le choix de l'outil, des techniques et des algorithmes, leur transposition au problème courant, et la modélisation de la solution et de son implémentation sont autant d'étapes-clés qui peuvent grandement aider à la production rapide et efficace d'*assets* techniques de qualité et réutilisables.

Par ce processus rigoureux, nous espérons gagner du temps tant sur le développement en lui-même que sur la maintenance du produit – car si entrer en production le plus vite possible donne un résultat plus rapide, ce temps gagné sur la pré-production risque d'être perdu en plus grandes quantités encore durant la maintenance.

L'intelligence artificielle est un cas intéressant pour explorer ce processus, étant donné son aspect abstrait et sa complexité théorique. C'est un aspect particulier du développement qui, à notre avis, peut grandement bénéficier de ce genre de démarche rigoureuse de conception, définition, recherche et documentation précises.

2.5 Conclusion

Au cours de ce chapitre introductif, nous avons mis en évidence les spécificités du monde de l'industrie du jeu-vidéo par rapport à l'industrie du logiciel « classique ».

Au travers des travaux de (Bethke 2003), nous avons décrit les compétences diverses et variées qui entrent en ligne de compte dans la production d'un logiciel vidéo-ludique. En particulier, nous avons distingué les rôles techniques, graphiques et de conception pour la partie développement de la production. Nous avons également mis en évidence le rôle de l'éditeur, en tant qu'investisseur dans le projet de développement.

Nous avons ensuite décrit les spécificités du cycle de développement type d'un jeu vidéo, à l'aide des travaux de (Bates 2004). Nous soulignons en particulier le rôle de la pré-production, étape entre l'idée d'origine pour le concept du jeu et la production dudit jeu, et qui consiste à définir tous les éléments qui composeront ce jeu, ainsi que sa planification.

Par après, nous avons montré les spécificités de l'intelligence artificielle appliquée au monde vidéo-ludique, grâce aux travaux de (Funge 2004). Nous mettons particulièrement en évidence la distinction entre *personnage-joueur* et *personnage non-joueur*, la structure de contrôleur pour ces *NPC*, et la nécessité de traiter chaque problème au cas par cas.

Enfin, au travers de l'expérience de (Bethke 2003), nous avons décrit quelques-uns des problèmes majeurs et récurrents que rencontrent les développements de logiciels vidéo-ludiques. En particulier, nous noterons la difficulté de générer du profit, la tension entre la pré-production et la production, et les défauts de planifications.

Face à ces problèmes, nous nous posons ensuite la question de ce que nous, en notre qualité de technicien, nous pouvons réaliser afin de réduire l'importance de ces problèmes. Nous avons alors émis l'avis de donner l'importance qu'il se doit à la pré-production, en réalisant un travail de conception et de documentation précise pour chaque aspect du développement technique.

Nous explorons cette idée à partir du chapitre suivant, où nous posons et tentons de résoudre un problème réel d'intelligence artificielle rencontrée lors de notre expérience dans l'industrie vidéo-ludique, et ce en tentant de pousser le travail de définition, modélisation, conception, spécification et documentation au plus loin.

Chapitre 3 – Etude de cas : recherche d'une solution

Au cours du chapitre précédent, nous avons mis en évidence les problèmes récurrents majeurs de l'industrie du jeu vidéo, à savoir principalement le manque de moyens, les défauts de planification et la sous-estimation de la période de pré-production.

En notre qualité de technicien, nous nous sommes alors intéressés aux qualités de la pré-production, et comment nous pouvions agir sur la réalisation d'un produit afin de gagner du temps et des moyens sur un développement courant, ainsi que les futurs.

Afin d'illustrer les avantages (et les inconvénients) d'une pré-production en profondeur telle que le suggérait Erik Bethke dans (Bethke 2003), nous allons donc mener l'étude d'un cas existant de l'industrie que nous avons rencontré lors de notre expérience « sur le terrain[18] ».

Nous nous mettons donc dans la peau du technicien qui reçoit un problème à résoudre de la part d'un *game designer*. Ce dernier pose les termes du problème et définit comment les solutions proposées seront évaluées.

Ce cas est d'abord étudié de manière abstraite et simplifiée. La solution, théorique, doit répondre à plusieurs contraintes : elle doit avoir un temps d'exécution le moins important possible tout en donnant le résultat le plus précis possible, et elle doit être implémentable facilement et rapidement.

La recherche de la solution, quant à elle, devra aller le plus en profondeur possible – c'est-à-dire, définir tous les aspects de la solution de manière ç ce que l'implémentation soit directe. Elle devra également être réalisée dans un temps le moins important possible, sans pour autant sacrifier de la profondeur.

3.1 Définition du problème

Dans le but de définir formellement le problème, nous allons poser l'environnement dans lequel nous évoluons – à savoir, le jeu, ses règles et son objectif. Nous définirons ensuite les données du problème, c'est-à-dire les variables, constantes et actions qui sont à notre disposition. Enfin, nous formulerons le problème de manière précise à l'aide de ces données.

3.1.1 Environnement

Le jeu que nous considérons est un simulateur géopolitique – c'est-à-dire une simulation reprenant les aspects politiques, économiques, militaires, scientifiques, culturels et sociaux des pays du monde réel, et les fait évoluer dans un monde virtuel.

[18] Notre expérience consiste en un stage de 6 mois en qualité de programmeur au sein de la société *Eversim*. Pour plus de détails concernant cette expérience, le lecteur pourra consulter le journal que nous avons tenu lors de ce stage dans l'annexe B.

Chaque pays est donc représenté au cours de cette simulation, qui capture les données géopolitiques du monde au 1er janvier 2011, et qui se termine le 1er janvier 2100. Les *NPC* les plus importants, ainsi que les *PC*, représentent les chefs d'état de ces différents pays.

Un chef d'état peut manipuler tous les aspects géopolitiques de son pays, en changeant par exemple le taux d'intérêt directeur de sa devise, en fixant le salaire minimum, en finançant la recherche militaire, en tentant d'espionner d'autres nations, en infiltrant des groupes terroristes, ou même en créant une journée nationale de la musique.

Le but d'un chef d'état est avant tout de rester au pouvoir – et pour ce faire, il doit rester populaire auprès de son peuple et auprès de son parti. Chaque action qu'il entreprend peut avoir un effet sur cette popularité, et une popularité trop faible lui fera perdre les prochaines élections, voire le fera destituer.

L'autre objectif d'un chef d'état est de développer son pays – ce qui passe par tenir un budget équilibré, surveiller l'inflation, combattre le chômage et stimuler la croissance. Il doit également être actif sur le plan militaire, en s'assurant de la défense de son pays, voire en en attaquant d'autres afin de s'approprier leurs ressources.

3.1.2 Données

Variables

Le premier élément auquel doit faire attention un chef d'état est sa **popularité**. Cette variable définit l'appréciation du peuple de son chef d'état, en termes de part de la population qui lui est favorable. Elle est exprimée en pourcentage, entre 0 et 100%.

En second lieu, le chef d'état doit faire attention à son **alignement politique**. En effet, un chef d'état est toujours issu d'un parti politique, qui possède une certaine orientation. Cette orientation est un entier entre -100 et 100, -100 étant l'extrême-gauche, 0 le centre strict, et 100 l'extrême-droite.

L'alignement politique d'un chef d'état est donc une valeur qui mesure son accord avec son parti. Au plus il fera d'actions en accord avec la politique de son parti, au plus cette valeur sera élevée. A contrario, s'il engage des actions qui vont à l'encontre des principes de son parti, cette valeur diminuera. Cet alignement est également un entier compris entre -100 et 100, -100 représentant le désaccord total, et 100 l'accord total.

Ensuite, le chef d'état doit surveiller les aspects économiques de son pays. Le plus important de ces aspects est le **déficit budgétaire**. Ce déficit est un réel qui représente la différence entre les recettes et les dépenses de l'état. Si ce nombre est négatif, on parle de déficit budgétaire. Si ce nombre est positif, on parle d'excédent budgétaire.

Pour parler du déficit, nous choisissons d'inverser l'opération, c'est-à-dire de considérer cette variable comme étant la différence entre les dépenses et les recettes de l'état. Un déficit positif correspondra donc à un déficit budgétaire, tandis qu'un déficit négatif correspondra à un excédent budgétaire.

Un second aspect économique est la **dette** d'un pays. La dette d'un pays est un réel dont le maximum est 0, et qui représente la quantité d'argent qu'un pays doit à ses créditeurs. La dette est mise à jour le 1er janvier de chaque année. Elle augmente si le déficit est positif, et diminue si le déficit est négatif.

Le *Produit Intérieur Brut*, ou **PIB**, est un autre aspect important de l'économie d'un pays. Le PIB est un réel qui est la mesure de la santé économique d'un pays, et représente la somme des produits de tous les secteurs d'activité de la nation.

La **croissance** représente l'évolution du PIB entre le 1er janvier d'une année et le 1er janvier de la suivante. Elle est exprimée en pourcentage, et est calculée comme suit :

$$Croissance = \left(1 - \left(\frac{PIB\ au\ 1er\ janvier\ de\ l'année\ n+1}{PIB\ au\ 1er\ janvier\ de\ l'année\ n}\right)\right) * 100 \qquad (3.1)$$

L'**inflation** représente l'évolution du niveau de prix des denrées et services au sein d'une nation. Tout comme la croissance, elle compare le niveau des prix au 1er janvier d'une année avec celui au 1er janvier de l'année suivante. Elle est exprimée en pourcentage, et est calculée comme suit :

$$Inflation = \left(1 - \left(\frac{Niveau\ des\ prix\ au\ 1er\ janvier\ de\ l'année\ n+1}{Niveau\ des\ prix\ au\ 1er\ janvier\ de\ l'année\ n}\right)\right) * 100 \qquad (3.2)$$

Le Tableau 2 récapitule les différentes variables, leur type, leur domaine et leur signification.

Variable	Type	Domaine	Signification
Popularité	Pourcentage	[0,100]	Part de la population qui apprécie le chef d'état
Alignement	Entier	[-100,100]	Alignement du chef d'état avec la politique de son parti
Déficit	Réel	$\Re$	Différence entre les dépenses et les recettes de l'état
PIB	Réel	$\Re$	Somme des produits des secteurs d'activité de la nation
Dette	Réel	]-∞,0]	Somme d'argent dont la nation est redevable auprès de ses créditeurs
Croissance	Pourcentage	[-100,+∞[	Rapport entre le PIB d'une année et le PIB de l'année suivante
Inflation	Pourcentage	[-100,+∞[	Rapport entre le niveau des prix d'une année et le niveau de l'année suivante

Tableau 2 : les variables de l'environnement

Notons que chaque variable possède deux versions : une valeur courante et une valeur estimée. La valeur courante est la valeur de la variable à l'instant où elle est observée. La valeur estimée est la valeur que la variable devrait atteindre au 1er janvier de l'année prochaine. Plus on se rapproche du 1er janvier, plus cette estimation se révèle précise.

Actions

Un chef d'état a à sa disposition une série d'actions qu'il peut réaliser. Ces actions ont pour la majeure partie un impact sur les différentes variables de l'environnement. Pour notre étude, nous excluons celles qui n'ont aucun impact sur les variables.

Tout d'abord, un chef d'état peut **modifier le budget, les effectifs ou les salaires** d'un ministère. Aussi, un chef d'état peut **créer, augmenter ou diminuer une taxe**. Il peut également **subventionner un secteur d'activité** pour le stimuler.

Enfin, pour soigner son image, un chef d'état peut **visiter un lieu public**, **assister à un événement populaire**, ou **donner une allocution à la télévision**.

3.1.3 Problème

Enoncé

Le problème consiste à réaliser un contrôleur pour les *NPC* chefs d'état, afin qu'ils contrôlent les aspects économiques de leur nation tout en restant populaires. L'objectif est donc de développer une **intelligence artificielle** qui s'assurera de **rester au pouvoir** tout en tentant **d'atteindre et de conserver un déficit nul ou négatif**.

Nous disposons donc d'un ensemble de variables qui doivent être observées - les variables de l'environnement - et d'un ensemble d'actions qui vont influer sur ces variables. A chaque **tour**, le contrôleur prendra donc en entrée les valeurs des variables, et produira en sortie un ensemble d'actions à exécuter.

Evaluation

La solution sera jugée satisfaisante si au 1er janvier 2050 de la simulation, 90% des pays ont un déficit budgétaire inférieur à 3% de leur PIB. Ces valeurs ont été choisies pour prendre en compte les possibles imprévus – l'évolution du monde étant incertain, particulièrement à long terme, nous ne sommes jamais à l'abri d'une absence de solution ou d'une solution dégradée juste à l'époque considérée.

Nous ne proposons aucun critère pour la popularité, celle-ci étant soumise à un bien trop grand nombre de facteurs externes que pour être contrôlée précisément. La solution doit donc prendre en compte cette variable dans sa prise de décision, mais n'en est pas responsable.

Aussi, la solution se devra d'être évolutive. Ajouter une action ou un indicateur au problème ne devrait en aucun cas remettre en cause l'implémentation. Enfin, la solution se devra de comporter un part de non-déterminisme, afin de s'assurer d'une dose minimale d'entropie.

Nous posons ces critères comme une **évaluation objective** de la solution – soit elle satisfait, soit elle ne satisfait pas un critère. Sur un total de 10 points, nous accordons 6 points au critère économique, car c'est l'objectif principal du contrôleur. Nous accordons alors 2 points au critère d'entropie et 2 points au critère d'évolutivité, vu qu'ils sont des objectifs secondaires.

Une solution sera donc notée sur 10, avec la note maximale correspondante à une implémentation acceptable et complète.

3.2 Modélisation de la solution

Le problème posé présente de nombreuses similitudes avec un problème d'optimisation : nous devons produire un choix parmi un ensemble d'actions, en fonction de leur impact sur des variables particulières, dont la valeur nous permet de mesurer l'évolution du problème.

Nous choisissons donc de ramener le problème posé à un **problème d'optimisation**. Dans un premier temps, nous allons modéliser le type de solution que nous allons rechercher, en suivant un canevas de problème d'optimisation.

Nous pourrons ensuite chercher et développer des solutions qui correspondront au type que nous aurons modélisé.

3.2.1 Fonction-objectif

Afin de réduire le problème donné à un problème d'optimisation, nous devons formaliser une fonction-objectif. Cette fonction-objectif doit évaluer l'état des variables d'environnement afin d'aiguiller le contrôleur dans son choix d'actions à réaliser.

La fonction-objectif comprendra alors autant de variables que de variables d'environnement considérées, pour lesquelles un ensemble d'actions qui influeront sur l'environnement seront à notre disposition.

Les variables de la fonction-objectif correspondent aux valeurs d'utilités des variables d'environnement considérées. Ces valeurs d'utilité sont des réels qui donnent une appréciation entre -100 et 100 des variables considérées. On nomme ces valeurs d'utilités **indicateurs**.

Chaque indicateur est donc le résultat d'une **fonction d'utilité** spécifique associée à la variable d'environnement qu'elle évalue.

Notons que certaines variables d'environnement, dont le déficit fait partie, se présentent sous deux formes : valeur courante et valeur estimée (voir 3.1.2). Nous choisissons de ne considérer que les valeurs estimées, car elles nous permettent d'adopter une meilleure vision à long terme.

La **fonction-objectif** prend la forme de la somme des valeurs des **indicateurs**, qui possèdent chacun une **pondération** réelle. En considérant n indicateurs $x_1 \dots x_n$, avec leurs pondérations respectives $a_1 \dots a_n$, la fonction-objectif peut être formalisée de la sorte :

$$Fitness = \sum_{j=1}^{n} a_j * x_j \qquad (3.3)$$

L'intérêt du contrôleur du chef d'état d'une nation est donc de produire des séquences d'actions qui **maximisent** la valeur de la fonction-objectif.

3.2.2 Données

Le **nombre d'actions à notre disposition** est formalisé par le naturel *m*, qui doit au moins valoir 1. Nous attribuons à chaque action possible un numéro identifiant – un naturel entre 1 et *n*. Par simplification, ce numéro correspond à l'emplacement de l'action dans la liste des actions à notre disposition.

La taille de la séquence d'actions que la solution devra produire en sortie est formalisée comme le naturel *s*, qui correspond au **nombre d'actions à exécuter** par tour et par contrôleur. Ce nombre d'actions doit être d'au moins 1.

La **séquence d'actions** à produire est donc une liste d'entiers notée *Choices*. Chaque élément de cette liste correspond au numéro d'une action. Nous faisons référence à la *kième* action choisie par cette notation : $Choices_k$,$1 \leq k \leq s$.

Nous aurons également besoin d'une copie de la valeur des variables d'environnement. Cette copie est formalisée par le **vecteur d'environnement** *V*, qui comporte alors *n* élément. On se réfère à la *jième* variable d'environnement par la notation V_j, $1 \leq j \leq n$.

Enfin, nous aurons besoin de connaître **l'impact de chaque action** sur les différentes variables d'environnement considérées. L'impact d'une action sur une variable d'environnement est un réel $impact_{ij}$ qui correspond à la variation théorique que l'action *i* infligera à la variable *j*. Cet impact est donc calculé de la sorte :

$$impact_{ij} = V'_j - V_j \qquad (3.4)$$

Le Tableau 3 reprend les différentes variables et constantes de notre modélisation, leur notation, leur type, leur domaine, et leur calcul si d'application.

Constante	**Notation**	**Type**	**Domaine**	
Nombre d'indicateurs	n	$\mathbb{N}$	$\mathbb{N}_0$	
Nombre d'actions considérées	m	$\mathbb{N}$	$\mathbb{N}_0$	
Nombre d'actions à choisir	s	$\mathbb{N}$	$[1,m]$	
Variable	**Notation**	**Type**	**Domaine**	**Calcul**
Fonction-objectif	$Fitness$	$\mathbb{R}$	$\mathbb{R}$	$Fitness = \sum_{j=1}^{n} a_j * x_j$
Indicateur	x_j	$\mathbb{R}$	[-100,100]	
Pondération d'un indicateur	a_j	$\mathbb{R}$	$\mathbb{R}^+$	
Séquence d'actions choisies	$Choices$	Liste de $\mathbb{N}$	Eléments : $[1,m]$ Taille : s	
Vecteur d'environnement	V	Liste de $\mathbb{R}$	Eléments : $\mathbb{R}$ Taille : n	
Impact d'une action	$impact_{ij}$	$\mathbb{R}$	$\mathbb{R}$	$impact_{ij} = V'_j - V_j$

Tableau 3 : Constantes et variables de la modélisation

3.2.3 Spécifications

Le type de solution que nous allons rechercher doit répondre aux spécifications de l'énoncé, et fera usage de notre modélisation de cet énoncé comme un problème d'optimisation.

En **entrée**, une solution prendra les valeurs des variables d'environnement considérées, ainsi que la liste des actions considérées à disposition du chef d'état. On lui renseignera également la nation du chef d'état ainsi que les pondérations a_j de cette nation pour les indicateurs considérés.

La solution définira alors une séquence d'actions à exécuter afin d'influer favorablement sur la fonction-objectif. Cette séquence formera la **sortie** de la solution.

Plus formellement, toute solution au problème modélisé répondra à ces spécifications :

Nom : - Solution au problème d'optimisation géopolitique
Input : - La liste des actions considérées à disposition d'un chef d'état. - La liste des valeurs des variables d'environnement du problème. - La nation à laquelle correspondent les variables d'environnement. - Les pondérations correspondant à l'importance accordée par la nation considérée à chaque variable d'environnement.
Output : - Une séquence de *s* numéros d'action, identifiant des actions à exécuter parmi les *m* actions disponibles au contrôleur.
Dépendances : - Une fonction d'utilité pour chacune des *n* variables d'environnement considérées par la solution. - Une fonction-objectif donnant une mesure à partir des valeurs d'utilités des variables d'environnement.

Spécification 1 : Solution au problème d'optimisation géopolitique

Les fonctions d'utilité associées aux variables d'environnement considérées répondront quant à elles à ces spécifications :

Nom : - Fonction d'utilité d'une variable d'environnement
Input : - Une variable d'environnement - La nation à laquelle correspond la variable d'environnement
Output : - Une valeur d'utilité mesurant la valeur de la variable d'environnement pour la nation considérée
Dépendances : - Aucune.

Spécification 2 : Fonction d'utilité d'une variable d'environnement

Enfin, la fonction-objectif répondra à ces spécifications :

Nom : - Fonction-objectif du problème d'optimisation géopolitique
Input : - La valeur d'utilité des *n* variables d'environnement considérées - La pondération de la valeur d'utilité de chacune des *n* variables d'environnement considérées
Output : - Une mesure correspondant à la somme des utilités pondérées.
Dépendances : - Aucune.

Spécification 3 : Fonction-objectif du problème d'optimisation géopolitique

3.3 Proposition de solutions

La première solution que nous envisageons est de **produire toutes les séquences possibles** de *s* actions parmi *m* discernables, et d'évaluer leur impact combiné sur les variables d'environnement. Cet impact modifierait la valeur de la fonction-objectif, et la meilleure séquence serait alors celle qui donnerait la meilleure modification – c'est-à-dire celle qui augmente le plus la valeur de la fonction-objectif.

Or, cette solution présente un défaut certain : explorer l'ensemble des solutions possibles en « force brute » présente une complexité algorithmique fortement exponentielle (de l'ordre de la taille de l'espace des solutions, c'est-à-dire $O(n^n)$).

Il existe cependant des heuristiques qui permettent d'approximer la solution en un temps moyen raisonnable. Etant donné la précision importante que donnerait une telle recherche, nous choisissons de développer cette solution malgré tout.

Nous envisageons cependant une seconde solution, qui se veut moins précise mais également moins coûteuse. Cette solution consiste en la décomposition de la recherche d'une séquence d'actions en **plusieurs sous-recherches**. Ainsi, pour produire une séquence de *s* actions, nous effectuerions *s* sous-recherches.
Une sous-recherche correspond à l'évaluation de l'impact de chaque action sur les variables d'environnement et leur impact sur la fonction d'utilité, afin d'établir un classement des actions – la meilleure étant celle qui augmente le plus la valeur de la fonction-objectif.

Il suffirait ensuite de poser un choix sur base de ce classement, de retenir l'impact de l'action choisie sur les variables d'environnement, et de réitérer le processus pour choisir une nouvelle action. Après avoir choisi *s* actions, la solution serait alors produite.

3.3.1 Exploration complète

La première solution que nous allons développer consiste donc à explorer l'ensemble des solutions possibles. Ce processus étant d'une complexité importante, nous choisissons [19] d'utiliser une heuristique s'appliquant aux cas d'optimisations : l'algorithme du **Tabou**.

Heuristique

Le principe de l'algorithme du Tabou, développé par Fred Glover en 1986, consiste à partir d'une solution générée aléatoirement, la **solution courante**. A partir de cette solution courante, l'algorithme génère un **voisinage**, c'est-à-dire des configurations de solution créées à partir de modifications atomiques de la solution courante.

L'algorithme choisit alors la **meilleure solution parmi le voisinage**, même si cette solution dégrade la fonction-objectif par rapport à la solution courante. L'algorithme génère ensuite un nouveau voisinage et réitère l'opération, jusqu'à atteindre la borne de la fonction-objectif ou le nombre maximum d'itérations.

L'algorithme retiendra quand même la meilleure solution découverte, toutes itérations confondues, afin de ne pas la perdre si toutefois la recherche s'aventurait dans la mauvaise direction.

La particularité de l'algorithme du Tabou est de tenter d'éviter les cycles et les optimums locaux en maintenant une liste de valeurs interdites, la **liste taboue**. Cette liste est mise à jour à chaque fois que l'on choisit une nouvelle solution, et empêche de choisir une nouvelle configuration si celle-ci comporte des valeurs interdites par la liste.

Il existe cependant une exception à cette règle. En effet, un **critère d'aspiration** peut être décrit, qui définit alors si une solution pourtant interdite peut tout de même être choisie. Par exemple, on choisira quand même une solution interdite si celle-ci est nettement meilleure que la meilleure solution déjà trouvée.

Notons également que la fonction-objectif associée à l'algorithme du Tabou doit être minimisée – une valeur moins importante est une valeur plus intéressante. Ainsi, si l'algorithme découvre une solution qui annule la fonction-objectif, il est certain d'avoir trouvé une des meilleures solutions envisageables, et peut alors s'arrêter.

[19] Nous avons connaissance de 3 types d'algorithme d'optimisation combinatoire : le Recuit Simulé, le Tabou, et les Algorithmes Génétiques. Il eut été préférable de comparer l'utilisation des 3 types et de choisir le meilleur – cependant, vu leur statut d'heuristique, leur comparaison théorique est relativement hasardeuse, leur efficacité dépendant du type de problème et de leur adaptation au problème.
Etant donné la contrainte de temps que nous avons, nous ne pouvons pas décemment modéliser et appliquer les 3 types d'algorithme. Nous faisons donc le choix d'utiliser celui qui, par expérience, a montré de meilleurs résultats que les autres : le Tabou.

Adaptation

Il nous faut adapter le principe de l'algorithme du Tabou à notre problème d'optimisation géopolitique.

Le premier élément à considérer est **la fonction-objectif**. Cette dernière nous pose problème, car notre modélisation nous donne une fonction-objectif à maximiser. Or, le Tabou demande une fonction-objectif à minimiser.

Nous n'avons malheureusement pas trouvé de transformation de cette fonction-objectif qui la rendrait minimisable. En effet, cela reviendrait à décrire la « solution parfaite », ce qui n'existe pas dans notre problème[20].

Nous faisons donc le choix de **ne pas considérer de borne à la fonction-objectif**, et donc de n'avoir pour seul critère d'arrêt le nombre maximum d'itérations permises.

Le deuxième élément à modéliser est la structure d'une solution. **Une solution correspond à une séquence d'actions**, telle que le contrôleur doit produire chaque tour. Elle est évaluée en combinant les impacts des actions qui la composent sur la fonction-objectif. **Le voisinage d'une solution est généré en modifiant des actions aléatoirement dans la séquence**.

La liste taboue, quant à elle, est structurée comme **une liste de couples comprenant le numéro d'une action et une position dans la séquence**. Si une solution générée reprend à la position donnée l'action associée, alors cette solution est interdite.

Nous choisissons de modéliser **le critère d'aspiration** comme la différence entre l'objectif associé à la meilleure solution découverte et l'objectif associé à la solution interdite considérée. Si cette différence dépasse un certain seuil, la solution est reprise malgré l'interdiction.

[20] Nous pourrions définir la solution parfaite comme une solution qui répond au moins au minimum des exigences sur les variables d'environnement, mais cela reviendrait à mettre sur un pied d'égalité un ensemble de solutions pourtant différentes – par exemple, une solution qui déboucherait sur un excédent budgétaire de 10% du PIB serait équivalente à une solution débouchant sur un excédent budgétaire de 2% du PIB. Nous désirons garder la première et non pas la seconde, ce qui nous empêche donc de définir une borne à la fonction-objectif.

Description

L'algorithme d'exploration de l'ensemble de l'espace des solutions commence par générer un ensemble de solutions construites aléatoirement. Il évalue ces solutions sur base de leur impact sur la fonction-objectif, et choisit la meilleure.

Sur base de cette première solution, l'algorithme va générer un voisinage, qui correspondra à un ensemble de solutions qui diffèrent d'un élément de la solution courante. De la même manière qu'il a choisi la première solution, l'algorithme considèrera la meilleure solution dans le voisinage, même si celle-ci dégrade la fonction-objectif par rapport à la solution courante. L'algorithme retient cependant en permanence la meilleure solution courante qu'il ait choisie.

Une fois la solution suivante choisie, il va retenir quel élément de la solution précédente a été modifié pour générer la nouvelle solution, et déclarera cet élément tabou. Si une solution générée par la suite comprend cet élément, elle sera déclarée interdite.

Une solution interdite peut toutefois être choisie, si son impact sur la fonction-objectif est nettement meilleur que celui de la meilleure solution retenue.

L'algorithme continue ainsi à générer un voisinage et à choisir une nouvelle solution jusqu'à atteindre le nombre d'itérations que nous lui avons fixé. A ce moment, il fournit la meilleure solution qu'il a retenue.

Modélisation

Le **nombre de solutions que l'algorithme génèrera à la base** est un naturel différent de 0 que nous noterons $NbrSolBase$.

La **taille du voisinage généré** autour d'une solution courante est également un naturel différent de 0 qui sera noté $TailleVoisinage$.

Le **nombre maximum d'itérations** que l'algorithme devra effectuer est noté $MaxIterations$. Ce nombre est un naturel différent de 0.

Le **seuil d'aspiration**, c'est-à-dire le seuil au-delà duquel la différence entre l'objectif de la meilleure solution courante et l'objectif d'une solution interdite permet à la solution interdite d'être choisie malgré tout, est noté *SeuilAspiration*. Ce seuil correspond à un réel positif.

La **taille de la liste taboue** définit le nombre de tabous que l'algorithme retiendra. Tout nouvel élément tabou ajouté alors que la liste a atteint sa taille maximale supprimera le dernier élément de la liste. Cette taille est un naturel différent de 0 et noté *TailleListeTabou*.

Une **liste de solutions** est une liste de séquences d'actions qui peuvent être potentiellement renvoyées comme choix du contrôleur pour un tour. Une telle liste est notée *ListeSolution*.

Un élément de la liste des tabous est nommé **couple tabou**, et est noté *CoupleTabou*. Ce couple tabou reprend deux éléments. Le premier est l'indice d'une action dans la liste des actions considérées. Le deuxième est un emplacement dans une séquence d'actions.

La **liste des tabous** est donc une liste de taille *TailleListeTabou* et comprenant des couples tabous.

Le Tableau 4 reprend les différentes constantes et variables de la solution, leur notation, leur type, et leur domaine.

Constante	**Notation**	**Type**	**Domaine**
Nombre de solutions de base générées	$NbrSolBase$	$\mathbb{N}$	$\mathbb{N}_0$
Taille du voisinage généré	$TailleVoisinage$	$\mathbb{N}$	$\mathbb{N}_0$
Nombre maximum d'itérations	$MaxIterations$	$\mathbb{N}$	$\mathbb{N}_0$
Seuil d'aspiration	$SeuilAspiration$	$\mathbb{R}$	$\mathbb{R}^+$
Taille de la liste Taboue	$TailleListeTabou$	$\mathbb{N}$	$\mathbb{N}_0$
Variable	**Notation**	**Type**	**Domaine**
Liste de solutions	$ListeSolution$	Liste de $Choices$	Eléments : $[1,m]$ Taille : $\mathbb{N}_0$
Couple tabou	$CoupleTabou$	$\mathbb{N} * \mathbb{N}$	$[1,m] * [1,k]$

Tableau 4 : Constantes et variables de la modélisation de la recherche complète

Exécution

L'exécution de l'algorithme proposé s'effectue alors suivant les étapes décrites ci-dessous.

1) Générer $NbrSolBase$ solutions de base aléatoirement.
2) Choisir la meilleure solution parmi les solutions générées.
3) Cette solution est maintenant la solution courante et la meilleure solution retenue.
4) Générer un voisinage de taille *TailleVoisinage* autour de la solution courante.
5) Si le voisinage est vide, passer à l'étape 12.
6) Choisir la meilleure solution dans le voisinage généré.
7) Comparer avec les éléments de la liste taboue. Si la solution reprise comprend un tabou et que son aspiration est inférieure à *SeuilAspiration*, la retirer du voisinage et revenir à l'étape 5.
8) Comparer la solution reprise avec la solution courante pour déterminer le couple tabou à rajouter à la liste taboue.
9) Ajouter le nouveau couple tabou à la liste taboue.
10) La solution courante est maintenant la solution reprise dans le voisinage.
11) Si la nouvelle solution courante est meilleure que la meilleure solution retenue, la meilleure solution retenue est alors la nouvelle solution courante.
12) Revenir à l'étape 4 jusqu'à ce qu'on ait effectué *MaxIterations* itérations.
13) Renvoyer la meilleure solution retenue.

Pseudo-code

Le pseudo-code décrit dans la figure Algorithme 1 reprend l'exécution de l'algorithme.

```
ListeSolutions Base = GénérerSolutionsDeBase(NbrSolBase);
Solution SolCourante = ChoisirMeilleureSolution(Base);
Solution MeilleureSol = SolCourante;
POUR i = 0 JUSQUE MaxIterations FAIRE
    ListeSolutions Voisinage = GénérerVoisinage(SolCourante, TailleVoisinage);
    Booléen SolTrouvée = Faux;
    TANT QUE ((~SolTrouvée) ET ~(Voisinage = Vide)) FAIRE
        Solution MeilleurVoisin = ChoisirMeilleureSolution(Voisinage);
        SI (SolutionTaboue(MeilleurVoisin, ListeTaboue)) ALORS
            SI (CritèreAspiration(MeilleurSol, MeilleurVoisin)) ALORS
                SolTrouvée = Vrai;
            SINON
                RetirerSolution(Voisinage, MeilleurVoisin);
            FIN SI
        SINON
            SolTrouvée = Vrai;
        FIN SI
    FIN TANT QUE
    SI (~(Voisinage = Vide)) ALORS
        CoupleTabou NouveauTabou = DeterminerTabou(SolCourante, MeilleurVoisin);
        AjouterTabou(ListeTaboue, NouveauTabou);
        SolCourante = MeilleurVoisin;
        SI (ImpactObjectif(SolCourante) > ImpactObjectif(MeilleureSol)) ALORS
            MeilleureSol = SolCourante;
        FIN SI
    FIN SI
FIN POUR
RENVOYER MeilleureSol;
```

Algorithme 1: Exploration complète

Spécifications

La spécification de l'algorithme correspond à celle du type de solution définie en Spécification 1 : Solution au problème d'optimisation géopolitique. Il reste cependant à spécifier les différentes sous-fonctions que l'algorithme utilise.

La première fonction correspond à *GénérerSolutionsDeBase*. Elle s'occupe de générer un nombre aléatoire de solutions, dont la quantité dépendra de son argument.

Nom : - Génération des solutions de base de l'algorithme
Input : - Le nombre de solutions à générer.
Output : - Une liste de solutions de taille correspondante au nombre donné en entrée. Une solution correspond à une instance de *Choices*.
Dépendances : - Un générateur aléatoire uniforme. (Spécification 13)

Spécification 4 : Génération des solutions de base de l'algorithme

La seconde fonction est une des plus critiques, car il s'agit de celle qui va choisir la meilleure solution parmi une liste de solutions. Elle correspond à *ChoisirMeilleureSolution*.

Nom : - Choix de la meilleure solution parmi une liste de solutions
Input : - Une liste de solutions.
Output : - Une solution qui correspond à la meilleure solution vis-à-vis de l'impact sur la fonction-objectif.
Dépendances : - Une fonction qui calcule l'impact d'une solution sur la fonction-objectif. (Spécification 12)

Spécification 5 : Choix de la meilleure solution parmi une liste de solutions

La fonction *GénérerVoisinage* s'occupe de générer le voisinage d'une solution courante. Elle créée donc un nombre renseigné de nouvelles solutions par modifications atomiques de la solution courante.

Nom : - Génération du voisinage d'une solution courante
Input : - Une solution correspondant à la solution courante. - La taille du voisinage à générer.
Output : - Une liste de solutions qui correspond au voisinage de la solution courante renseignée, et dont la taille équivaut au nombre renseigné.
Dépendances : - Une fonction qui génère une nouvelle solution à partir d'une solution courante. (Spécification 14)

Spécification 6 : Génération du voisinage d'une solution courante

La fonction *SolutionTaboue* définit si une solution proposée comporte un tabou repris dans la liste des tabous.

Nom : - Vérification de l'interdiction d'une solution
Input : - Une solution correspondant à la solution considérée. - La liste des couples tabous.
Output : - Vrai si la solution renseignée est interdite, Faux sinon.
Dépendances : - Aucune.

Spécification 7 : Vérification de l'interdiction d'une solution

La fonction *CritèreAspiration* définit si une solution proposée est tellement supérieure à la meilleure solution actuellement considérée, qu'elle ne peut être la cible d'une interdiction sur base de la liste des tabous.

Nom : - Vérification du critère d'aspiration d'une solution.
Input : - Une solution correspondant à la solution considérée. - La meilleure solution actuellement retenue.
Output : - Vrai si la solution considérée vérifie le critère d'aspiration et donc ne peut être interdite ; Faux sinon.
Dépendances : - Une fonction qui calcule l'impact d'une solution sur la fonction-objectif. (Spécification 12)

Spécification 8 : Vérification du critère d'aspiration d'une solution

La fonction *RetirerSolution* enlève une solution d'une liste de solutions.

Nom : - Suppression d'une solution d'une liste de solutions.
Input : - La liste de solutions. - La solution à enlever de la liste de solutions.
Output : - Aucun.
Dépendances : - Aucune.

Spécification 9 : Suppression d'une solution d'une liste de solutions.

La fonction *DeterminerTabou* calcule un nouveau couple tabou à partir d'une solution courante et d'une nouvelle solution qui ne diffère que d'un élément.

Nom : - Calcul d'un nouveau couple tabou
Input : - Une solution correspondant à la solution courante. - Une solution correspondante à la nouvelle solution considérée et qui ne diffère que d'un élément de la solution courante.
Output : - Le couple tabou qui correspond à l'élément de la solution courante qui a été modifié et de son emplacement dans la solution.
Dépendances : - Aucune.

Spécification 10 : Calcul d'un nouveau couple tabou

La fonction *AjouterTabou* ajoute un nouveau couple tabou à la liste des tabous. Si la liste dépasse alors la taille qui lui a été donnée, le dernier élément de la liste (donc le plus ancien) est oublié.

Nom : - Ajout d'un nouveau couple tabou à la liste des tabous
Input : - La liste des tabous. - Le couple tabou à ajouter à la liste des tabous.
Output : - Aucun.
Dépendances : - Aucune.

Spécification 11 : Ajout d'un nouveau couple tabou à la liste des tabous

La fonction ImpactObjectif définit l'impact d'une solution sur la fonction-objectif.

Nom : - Calcul de l'impact d'une solution sur l'objectif
Input : - Une solution correspondant à la solution considérée.
Output : - La différence entre la valeur estimée de la fonction-objectif après exécution de l'action et la valeur courante.
Dépendances : - Une fonction d'utilité par variable d'environnement considérée par la fonction-objectif. (Spécification 2) - Une fonction calculant la valeur de la fonction-objectif. (Spécification 3) - Une fonction calculant l'impact d'une action sur les variables d'environnement. (Spécification 15 : Calcul de l'impact d'une action sur l'environnement

Spécification 12 : Calcul de l'impact d'une solution sur l'objectif

Les précédentes spécifications ont amené à découvrir trois nouvelles fonctions. La première correspond à un générateur aléatoire uniforme.

Nom : - Génération d'une valeur aléatoire uniforme
Input : - La borne supérieure pour la valeur aléatoire.
Output : - Une valeur réelle aléatoire entre 0 et la borne supérieure spécifiée. La génération doit être uniforme, c'est-à-dire que chaque valeur a la même probabilité d'être tirée.
Dépendances : - Aucune.

Spécification 13 : Génération d'une valeur aléatoire uniforme

La seconde fonction mise en évidence est une fonction qui calcule un voisin à partir d'une solution courante. Ce voisin ne diffèrera que d'un seul élément par rapport à la solution courante.

Nom : - Génération d'un voisin d'une solution courante.
Input : - La solution courante.
Output : - Une nouvelle solution qui ne diffère que d'un élément de la solution courante.
Dépendances : - Un générateur aléatoire uniforme. (Spécification 13)

Spécification 14 : Génération d'un voisin d'une solution courante.

La troisième fonction correspond au calcul de l'impact d'une action sur les variables d'environnement.

Nom : - Calcul de l'impact d'une action sur l'environnement
Input : - Le numéro de l'action choisie. - La liste des variables d'environnement considérées.
Output : - La liste des variables d'environnement est mise à jour avec l'impact de l'action choisie.
Dépendances : - Aucune.

Spécification 15 : Calcul de l'impact d'une action sur l'environnement

Complexité

La complexité théorique en temps de l'algorithme de recherche complète va dépendre de son implémentation – et principalement de l'implémentation de ses sous-fonctions. Nous pouvons cependant estimer une borne supérieure.

Notons avant tout que lorsque nous parlerons de complexité théorique en temps, nous considérons le pire des cas (*Worst-case scenario*). Et vu que nous produisons des estimations, nous tentons de donner une borne supérieure[21] de complexité, c'est-à-dire que nous tentons d'estimer le nombre d'opérations requises pour mener à bien la spécification en « force brute ».

Nous estimons la complexité du générateur aléatoire d'être en *O(n),* du fait qu'il ne dépendrait que de l'intervalle de valeurs qui lui est donné. De ce fait, *GénérerSolutionsDeBase,* qui dépendrait du nombre de solutions à générer, serait de complexité $O(n^2)$ – $O(n)$ solutions par $O(n)$ générations. *GénérerVoisinage*, pour les mêmes raisons, serait en $O(n^2)$.

Le calcul de *ImpactObjectif* serait lui en $O(n)$ car il ne dépendrait que du nombre de variables d'environnement considérées. *ChoisirMeilleureSolution* devrait appliquer *ImpactObjectif* sur toutes les solutions à évaluer, et serait donc en $O(n^2)$.

SolutionTaboue et *AjouterTabou* devraient au pire parcourir toute la liste des tabous, et dépendrait donc uniquement de la taille de la liste. Leur complexité serait donc en $O(n)$. De même, *RetirerSolution* ne parcourra qu'au pire l'ensemble de la liste des solutions, et ne dépendrait que de la taille de cette liste. Elle serait donc également de complexité $O(n)$.
CritèreAspiration ne ferait qu'appliquer deux fois le calcul de la fonction-objectif, qui serait en $O(n)$. Elle resterait donc de complexité $O(n)$. Enfin, *DeterminerTabou* ne dépendrait que de la taille des séquences d'actions qui forment les solutions, car il suffirait de parcourir les deux solutions deux à deux, élément par élément. Elle serait donc complexité $O(n)$.

La partie la plus complexe de l'algorithme serait donc les deux boucles imbriquées des lignes 4 à 27 et 7 à 18. Ces deux boucles dépendent respectivement du nombre maximum d'itérations et de la taille du voisinage d'une solution, qui, imbriquées, donnerait une complexité de $O(n^2)$.

Or, *ChoisirMeilleureSolution* se trouve à l'intérieur de la deuxième boucle imbriquée. Cette fonction étant de complexité $O(n^2)$ et étant appliquée au pire de l'ordre de $O(n^2)$ fois, la borne supérieure de complexité de l'algorithme serait alors $O(n^4)$.

[21] Une borne supérieure de complexité peut sembler ne pas avoir de sens, vu que l'on peut développer des solutions infiniment complexes s'il nous en prenait l'envie. Nous posons simplement ici comme borne supérieure l'approche en « force brute », c'est-à-dire un algorithme qui ne chercherait à donner la solution en la calculant directement, sans chercher à passer par des optimisations. Un exemple type d'algorithme à « force brute » est un algorithme de *Generate & Test* : générer toutes les solutions possibles, et les tester pour trouver la meilleure ou tout du moins une qui fonctionne.

Par rapport à la complexité théorique correspondant à générer l'ensemble des solutions et à choisir la meilleure, qui serait de l'ordre de $O(n^n)$, nous avons donc économisé une partie relativement importante de complexité, au prix d'une perte de précision.

3.3.2 Décomposition en sous-recherches

La deuxième solution que nous allons explorer se veut plus simple que la première, bien que moins précise. Au lieu de considérer l'ensemble des solutions possibles, nous décomposons la recherche de la solution en plusieurs sous-recherches, en gardant à chaque étape la meilleure action.

Notre espoir quant aux résultats de cet algorithme, est que les sous-recherches dirigent la recherche globale – en ne considérant que les meilleures actions à chaque étape. Nous risquons peut-être de n'obtenir que des optimums locaux, mais nous espérons que ces optimums seront suffisamment intéressants que pour satisfaire aux exigences.

Description

L'algorithme de décomposition en sous-recherches va garnir élément par élément la séquence d'actions qui compose la solution. Le principe est donc de choisir une action, de la placer à la première place libre de la séquence, de retenir son impact sur les variables d'environnement, et de recommencer ainsi jusqu'à ce que la séquence soit complète.

Pour choisir une action, nous allons calculer l'impact de chaque action disponible sur la fonction-objectif. Cet impact consistera en la différence entre la valeur qu'aurait la fonction-objectif si on appliquait l'action, et la valeur courante de la fonction.

Ensuite, nous trierons les actions par ordre décroissant d'impact – la meilleure action est donc celle classée en premier lieu. Cependant, si nous considérions toujours la meilleure action d'office, nous aurions bien peu d'entropie.

En effet, les paramètres de jeu étant constants au lancement, à moins d'événements externes aléatoires importants, les décisions devraient rester relativement les mêmes d'une partie à l'autre.

Pour parer à ce problème, nous allons choisir aléatoirement une action parmi celles considérées, en donnant plus de poids aux meilleures et moins de poids au moins bonnes.

Pour ce faire, nous allons générer une valeur entre 0 et la somme des poids positifs. Ensuite, nous parcourrons la liste des actions dans l'ordre décroissant des poids, en retranchant le poids de chaque action à la valeur générée. Une fois cette valeur négative ou nulle, ou une fois que l'on rencontre des poids négatifs ou nuls, nous choisissons l'action sur laquelle nous sommes arrivés.

Il suffit ensuite de conserver l'impact de l'action choisie sur les variables d'environnement, et de reprendre la recherche, autant de fois qu'il le faudra pour garnir la séquence d'actions qui compose la solution.

Modélisation

Le **poids d'une action** correspond à la différence entre la valeur qu'aurait la fonction-objectif si on appliquait l'action, et la valeur courante de la fonction. Il est positif si l'action améliore l'objectif, nul si elle est indifférente, et négative si elle dégrade l'objectif.

La **liste des actions et de leur poids** sera notée *PoidsActions*. Un élément de cette liste est un couple qui comprend le numéro d'une action et son poids. Cette liste devra être triée par ordre décroissant de poids, ce qui formera le classement des actions.

Afin d'assurer une part d'entropie, nous ne choisirons pas systématiquement l'action avec le poids le plus important. Nous calculerons un **arbitre**, noté $Arbitre$, et qui correspondra à une valeur réelle aléatoire entre 0 et la somme des poids positifs.

Nous parcourrons alors la liste des actions dans l'ordre du classement, en retranchant à l'arbitre le poids de chaque action. L'action qui rendra la valeur de l'arbitre négative ou nulle sera l'action choisie par le contrôleur.

Notons cependant notre choix de ne considérer que les poids positifs. En effet, un poids négatif indiquerait une dégradation de la fonction-objectif, ce qui n'est pas dans l'intérêt de cette recherche. Nous arrêtons donc la sélection d'une solution avant de considérer les poids négatifs.

Le Tableau 5 reprend la liste des variables qui interviennent dans notre modélisation.

Variable	Notation	Type	Domaine
Liste des actions et de leurs poids	*PoidsActions*	Liste de $\mathbb{N} * \mathbb{R}$	Eléments : $[1, m] * \mathbb{R}$ Taille : m
Arbitre	$Arbitre$	$\mathbb{R}$	$\mathbb{R}^+$

Tableau 5 : Variables de la modélisation de la décomposition en sous-recherches

Exécution

L'exécution de l'algorithme de décomposition en sous-recherches suit les étapes décrites ci-dessous.

1) Evaluer l'impact de chaque action sur la fonction-objectif
2) Construire la liste des actions et de leur poids sur base des impacts
3) Classer la liste des actions par ordre décroissant de poids
4) Générer un arbitre avec une valeur entre 0 et la somme des poids positifs
5) Parcourir la liste dans l'ordre décroissant du classement, en réduisant l'arbitre par le poids de chaque action rencontrée
6) S'arrêter lorsque l'arbitre est négatif ou nul, ou que la prochaine action a un poids négatif.
7) Insérer l'action sur laquelle l'algorithme s'est arrêté dans la séquence d'actions composant la solution
8) Retenir l'impact de l'action choisie sur les variables d'environnement
9) Retourner à l'étape 1 jusqu'à ce que la séquence d'actions composant la solution soit remplie
10) Renvoyer la séquence d'actions ainsi produite.

Pseudo-code

Le pseudo-code décrit dans la figure Algorithme 2 reprend l'exécution de l'algorithme.

```
1   PoidsActions Poids;
2   Double SommePoids;
3   POUR i = 0 JUSQUE s FAIRE
4       EvaluerImpactActions(Poids, SommePoids);
5       Tri(Poids);
6       Double Arbitre = GénérerArbitre(SommePoids);
7       Entier ActionChoisie = Poids[0].indice;
8       Entier RangAction = 1;
9       TANT QUE ((Arbitre > 0) ET (Poids[RangAction].poids > 0)) FAIRE
10          ActionChoisie = Poids[RangAction].indice;
11          Arbitre -= Poids[RangAction].poids;
12          ++RangAction;
13      FIN TANT QUE
14      Solution[i] = ActionChoisie;
15      SauverImpact(ActionChoisie, ListeVariablesEnv);
16  FIN POUR
```

Algorithme 2 : Décomposition en sous-recherches

Spécifications

La spécification de l'algorithme correspond à celle du type de solution définie en Spécification 1 : Solution au problème d'optimisation géopolitique. Il reste cependant à spécifier les différentes sous-fonctions que l'algorithme utilise.

La fonction *EvaluerImpactActions* construit la liste des actions et de leur poids, tout en calculant la somme des poids positifs.

Nom : - Evaluation de l'impact des actions considérées
Input : - La liste des actions et de leur poids qui sera garnie par la fonction. - La variable devant contenir la somme des poids positifs et qui sera garnie par la fonction.
Output : - La liste des actions et de leur poids a été garnie. - La variable devant contenir la somme des poids a été garnie.
Dépendances : - Une fonction d'utilité par variable d'environnement considérée par la fonction-objectif. (Spécification 2) - Une fonction calculant la valeur de la fonction-objectif. (Spécification 3) - Une fonction calculant l'impact d'une action sur les variables d'environnement. (Spécification 15 : Calcul de l'impact d'une action sur l'environnement

Spécification 16: Evaluation de l'impact des actions considérées

La fonction *Tri* s'occupe de trier la liste des actions et de leur poids par ordre décroissant de poids.

Nom : - Tri de la liste des actions et de leur poids
Input : - La liste des actions et de leur poids.
Output : - La liste des actions et de leur poids a été triée par ordre décroissant de poids.
Dépendances : - Aucune.

Spécification 17 : Tri de la liste des actions et de leur poids

La fonction *GénérerArbitre* construit la valeur de l'arbitre sur base de la somme des poids.

Nom : - Génération de l'arbitre
Input : - La somme des poids positifs des actions considérées.
Output : - Une valeur générée aléatoirement entre 0 et la somme des poids renseignée.
Dépendances : - Un générateur aléatoire uniforme. (Spécification 13)

Spécification 18 : Génération de l'arbitre

Enfin, la fonction *SauverImpact* porte l'impact d'une action choisie sur la copie des variables d'environnement qu'utilise l'algorithme. Cela correspond à la Spécification 12.

Complexité

Tout comme la première solution, nous allons estimer la complexité théorique en temps de l'algorithme, et ce en considérant le pire des cas (*Worst-case scenario*). Rappelons donc que lorsque nous parlons de complexité, nous faisons référence à la complexité en temps dans le pire des cas.

Dans un premier temps, nous allons d'abord estimer la complexité des différentes sous-fonctions, en donnant leur borne supérieure de complexité, qui correspond à une approche « force brute ».

La fonction *EvaluerImpactActions* parcourt la liste de toutes les actions et considère leur impact sur tous les indicateurs. Ces deux dimensions lui valent une complexité estimée en $O(n^2)$.

La fonction *Tri*, en force brute, donnerait une complexité en $O(n^2)$. Cela correspondrait à rechercher le poids maximum, de l'amener en tête, et de reprendre ainsi jusqu'à ce que toute la liste ait été triée.

La fonction *GénérerArbitre* ne dépend que du générateur aléatoire, qui comme nous l'avons estimé précédemment possède une complexité en $O(n)$.

Enfin, *SauverImpact* applique une modification sur la liste complète des variables d'environnement. Son unique dimension lui vaut donc une complexité estimée en $O(n)$.

L'algorithme de décomposition en sous-recherches comporte deux dimensions, deux boucles imbriquées : le nombre d'actions à produire pour la séquence d'actions composant la solution, et le choix d'une action dans la liste à l'aide de l'arbitre.

Les fonctions *EvaluerImpactActions* et *Tri* dépendent toutes deux de la dimension du nombre d'actions à produire. Elles-mêmes en $O(n^2)$, elles portent la complexité la plus importante de l'algorithme à $O(n^3)$.

3.4 Conclusion

Notre étude de cas nous a d'abord mené à donner le contexte du problème : l'environnement, les données, l'énoncé et comment une solution sera évaluée. Fort de ces ressources et de ces exigences, nous avons modélisé une solution type au problème.

De cette solution-type, nous avons tiré deux solutions particulières : une qui considère l'ensemble des solutions, et une autre qui décompose la recherche de la solution en plusieurs sous-recherches.

La première solution serait trop complexe à mettre en œuvre si on devait calculer toutes les solutions possibles. Nous avons donc fait le choix d'utiliser une heuristique pour parcourir cet espace de solution : l'algorithme du *Tabou*.

Cette solution nous a permis d'atteindre une complexité relativement moins importante, mais cela nous a coûté de la précision.

La seconde solution est beaucoup plus simple à mettre en œuvre, mais ignore un grand nombre de solutions possibles. En considérant la meilleure solution pour chaque action à choisir, nous espérons cependant aiguiller la recherche vers un optimum suffisamment efficace que pour satisfaire aux exigences.

Le chapitre suivant décrit le produit sur lequel ces solutions seront déployées, et montre comment elles seront implémentées.

Chapitre 4 - Etude de cas : application de la solution

Maintenant que nous avons défini formellement le problème et nos propositions de solutions pour le résoudre, il nous faut décrire comment ces solutions seront implémentées dans le logiciel cible : *Rulers of Nations*.

Tout d'abord, nous présenterons l'environnement dans lequel est développé le logiciel, à savoir le studio de développement *Eversim*. Nous décrirons ensuite son architecture logique et ses principales structures de données.

Enfin, nous reprendrons chaque spécification que nous avons définie afin de montrer comment elles seront implémentées au sein de ce logiciel.

Notons que toute description du logiciel-cible, *Rulers Of Nations*, doit être considérée sous couvert de la confidentialité, ses éléments appartenant à la société *Eversim*.

4.1 Environnement

Afin de remettre le logiciel dans son contexte, nous présentons l'environnement dans lequel il évolue. Tout d'abord, nous présentons le studio de développement derrière *Rulers of Nations*. Nous rappellerons ensuite les principales règles du jeu.

Enfin, nous décrirons l'implémentation actuelle du jeu en présentant son architecture logique et ses principales structures de données.

4.1.1 Studio de développement

Afin de présenter le studio de développement responsable de la production de *Rulers of Nations*, nous nous permettons de citer la description que l'entreprise renseigne sur son site Internet[22] :

« La société Eversim est spécialisée dans la création et le développement de jeux de simulation, de stratégie, et de rôle, fonctionnant en réseau et en univers persistant. Ses produits sont destinés au grand public, ainsi qu'aux professionnels des secteurs de la défense ou de la sécurité pour l'apprentissage et l'entrainement (Serious Gaming).

Le développement de ces produits innovants est notamment soutenu par le OSEO, la Direction Générale de l'Armement (DGA) et le CNC.

La société Eversim S.A. a été créée en février 2004 par trois associés, André et Louis-Marie Rocques et Pascal Einsweiler. L'entreprise est implantée à Lognes (Marne-La-Vallée) à l'Est de Paris et est soutenue financièrement par un groupe d'investisseurs d'Ile-de-France présent dans le capital de l'entreprise.

Les fondateurs d'Eversim ont travaillé plus de vingt-cinq ans dans l'univers du jeu vidéo et étaient déjà présents aux balbutiements de cette industrie au début des années 1980.

[22] EVERSIM. *Eversim : Simulation and Serious Games* [en ligne]. Disponible sur : <http://www.eversim.com/ns/fr/corporate.php>. (page consultée le 10 février 2011).

Ils ont créé entre 1987 et 2003, via le label Silmarils, près d'une trentaine de produits pour consoles et ordinateurs (PC, PS2, PS1, Imac, etc...) qui se sont vendus dans le monde entier. »

4.1.2 Règles du jeu

Le logiciel *Rulers of Nations* est le deuxième opus d'un jeu de simulation géopolitique, *Geopolitical Simulator* ou *GPS*. Les points clés de ce produit sont repris sur le site Internet qui lui est dédié[23] :

« Rulers of Nations est une simulation géopolitique de notre monde actuel. Les joueurs y incarnent des chefs d'État ou de gouvernement (président, roi, premier ministre…) d'un pays qu'ils choisissent au départ, et peuvent agir dans de très nombreux domaines : économique, social, militaire, politiques intérieure et étrangère, environnemental, culturel,etc.

Tous les pays du monde sont représentés avec leurs variables et leur fonctionnement propres. Le jeu regroupe des phases de gestion économique, de commerce, de wargame, de construction, d'espionnage, de simulation et de manipulation politique.

Deux modes de jeu sont proposés :

- un mode *Compétition*, jouable en solo ou en multijoueur, qui opposent seize grandes nations (jouées par l'Intelligence Artificielle ou par d'autres joueurs humains). Dans ce mode, tous les 'coups' sont permis pour devenir la première nation du monde. Les joueurs ont la possibilité de choisir des objectifs secrets, qui augmenteront leur score s'ils sont atteints. Un classement des joueurs en ligne, ainsi que la liste des parties en cours sont accessibles sur ce site.
- un mode *Simulation*, jouable en solo uniquement, dans lequel le joueur dirige une des 170 nations proposées. Une vingtaine de scénarios contextuels avec l'actualité du monde d'aujourd'hui sont disponibles comme par exemple 'Sortie de crise mondiale', 'Afghanistan : nouveau Vietnam ?' ou 'Menaces terroristes aux JO de Londres 2012 !'.

Sur le fond, le joueur-chef de l'État a toujours comme objectif de rester populaire, notamment lors des élections, afin de durer le plus longtemps possible au pouvoir. Le maintien des comptes de l'État et du déficit budgétaire est aussi un élément important pour sa longévité.

Le joueur peut intervenir dans les menus de gestion, sur la carte, lors des rendez-vous et en répondant aux diverses requêtes et alertes.

- les menus de gestion des ministères : plusieurs centaines d'actions et législations sont disponibles. Exemples : répartition du budget entre les différents postes budgétaires, modification du taux de TVA, rallongement de l'âge de la retraite, création d'une nouvelle taxe, limitation des pouvoirs des partis politiques, augmentation des minimas sociaux, changement du mandat du chef de l'État.

[23] EVERSIM. *Rulers of Nation* [en ligne]. Disponible sur : <http://www.rulers-of-nations.com/presentation.php?langue=fr>. (page consultée le 10 février 2011).

Chaque proposition de loi doit être votée au Parlement - pour les régimes démocratiques s'entend -, et celui-ci se détermine en fonction des votes des différents partis politiques présents (qui sont conformes à la réalité).

Le joueur peut aussi agir directement, sans l'aval du Parlement, comme par exemple pour nommer les ministres de son gouvernement, visiter un hôpital, faire une intervention télévisée, fixer son programme électoral, ordonner une enquête des services secrets.

Ceux-ci jouent un rôle prépondérant dans le jeu, ils peuvent permettre en autres de connaître les forces militaires d'un pays, d'espionner un parti politique, de lancer des scandales, de faire assassiner un opposant, de mener des opérations de sabotage.»

Le principal aspect du jeu qui intéresse notre recherche est le mode *Simulation*. En effet, il existe déjà une intelligence artificielle pour le mode *Compétition* qui détermine des actions à effectuer pour les pays compétiteurs non-joueurs.

Afin de ne pas interférer avec cette IA, nous décidons de n'appliquer la solution qu'au mode *Simulation*, celui-ci ayant de plus la caractéristique de durer bien souvent plus longtemps.

Nous remarquons que les règles et objectifs qui intéressent particulièrement notre modélisation sont repris dans la section 3.1 (Définition du problème).

4.1.3 Architecture logique

Le diagramme de composants repris en Figure 5 décrit l'architecture logique du logiciel *Rulers of Nations*. Cette architecture est composée de plusieurs composants majeurs, orientés autour du moteur principal.

Moteur principal

Le **moteur principal** est le composant central de l'architecture de *Rulers of Nations*. Ce composant coordonne les événements et exécutions de tous les autres composants du logiciel.

Ce composant est également responsable du fonctionnement de plusieurs sous-systèmes globaux. Le premier de ces sous-systèmes est le **gestionnaire des lois**. Ce gestionnaire régit la valeur courante des lois des différents pays, et s'occupe de coordonner la prise en charge et le vote de nouvelles propositions de loi.

Présent également, **le gestionnaire des catastrophes**. Ce gestionnaire prend en charge la création et le fonctionnement des catastrophes qui peuvent apparaître tout au long du jeu – cyclones, incendies, accidents nucléaires, …

Le **gestionnaire des événements** s'occupe des *déclencheurs*. Un déclencheur est un événement qui peut être accompagné d'une date définissant quand il devra être exécuté. Cet événement fait référence à une réaction de la base des réactions, et le gestionnaire s'occupe de l'interpréter dans le jeu.

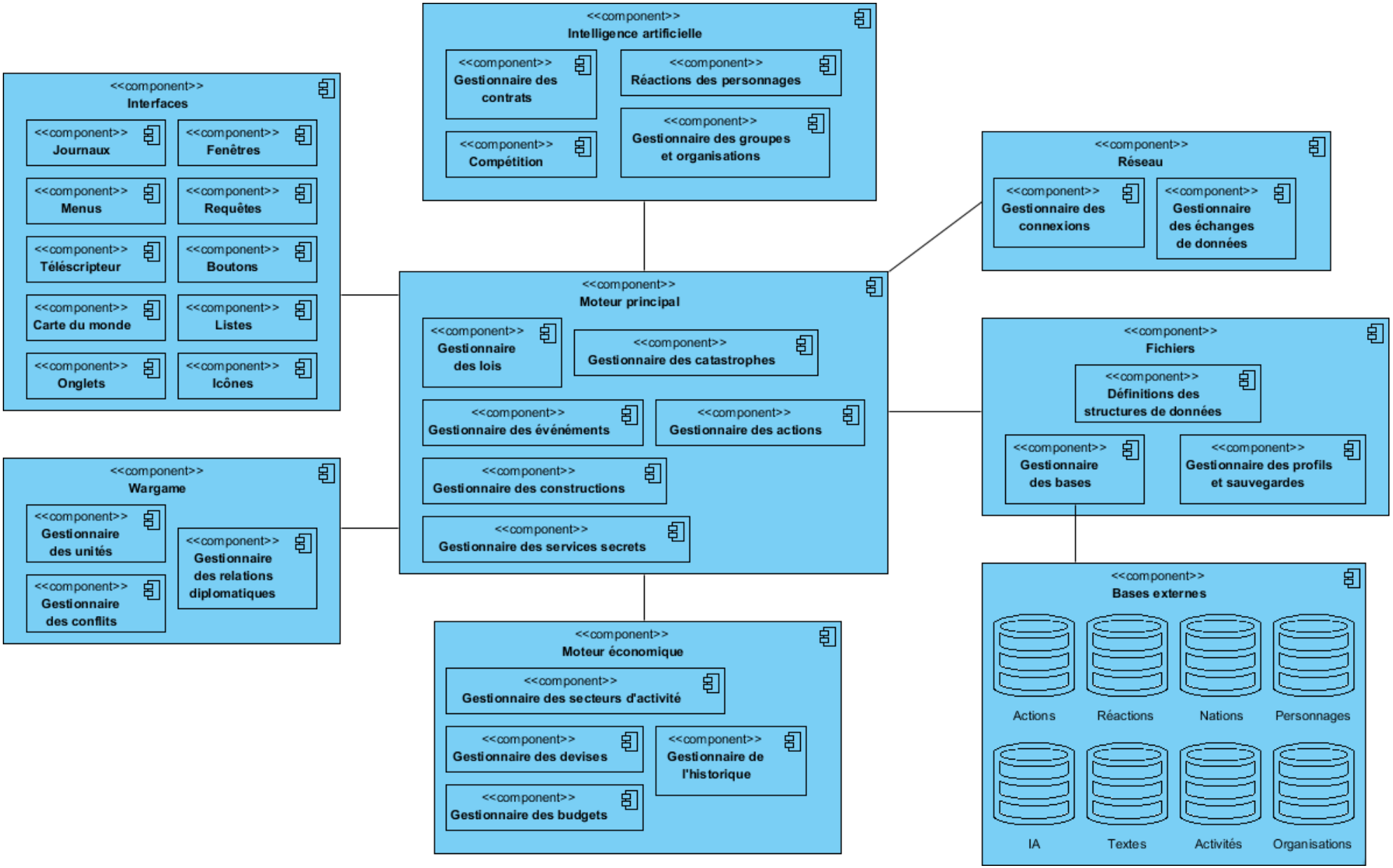

Figure 5 : Architecture logique de Rulers of Nations

Le **gestionnaire des actions** s'occupe de récupérer les différentes actions engagées par les chefs d'état, et de les exécuter. Ce gestionnaire peut également estimer le coût ou le profit qu'une action génèrera, si cela est applicable, afin de renseigner un joueur sur les conséquences de son action.

Le **gestionnaire des constructions** récupère les demandes de nouvelles constructions, de réparations ou de démantèlement, calcule leur coût, gère leur positionnement géographique si applicable, et mène le processus de construction à bien.

Enfin, le **gestionnaire des services secrets** s'occupe du fonctionnement des services secrets de chaque nation. Ces services secrets recoupent l'espionnage et le contre-espionnage, les enquêtes, et les assassinats.

Interfaces

Les **interfaces** sont gérées indépendamment du code *business*. Elles récupèrent les informations de la part des autres composants et les présentent au joueur.

Nous retrouvons parmi les éléments d'interface des composants très génériques comme des **fenêtres**, des **boutons**, des **listes**, des **icônes** et des **menus**. Ces éléments peuvent héberger et renvoyer des informations, présentées avec une police, des couleurs, des sons, des animations, des textures et des modèles définis.

Les interfaces comportent également des sous-systèmes gérant des aspects plus spécifiques. Le premier de ces sous-systèmes est la **carte du monde**, système critique s'il en est. La carte du monde s'occupe d'afficher, comme son nom l'indique, le monde sous forme 2D ou 3D. Sur la carte du monde, nous retrouvons diverses informations comme les pays, les troubles, et les conflits et catastrophes en cours, ainsi que les unités comme les villes et les armées.

Chaque lundi de la simulation, un **journal** reprenant les événements importants de la semaine précédente est construit. Un sous-système des interfaces s'occupe de cette construction et de son affichage au joueur.

Tout au long du jeu, un joueur recevra des **requêtes** de la part de différents acteurs – personnages comme organisations. Ces requêtes peuvent être simplement informatives, ou demander une action de la part du joueur. Un sous-système des interfaces s'occupe de construire et d'afficher ces requêtes.

De même, un **téléscripteur** présente de manière concise les grands événements en cours dans la nation d'un joueur et dans le monde en général. Ces événements recoupent les grèves, manifestations et autres troubles, les conflits, les catastrophes, et les signatures de traités et de contrats économiques.

Enfin, un sous-système des interfaces s'occupe de la construction et de la présentation des **onglets**. Un onglet est une partie d'une fenêtre qui comporte des éléments spécifiques définis dans les bases. Généralement, il s'agit de présenter des informations sur un thème précis et de proposer des actions par rapport à ce thème.

Intelligence artificielle

La gestion de **l'intelligence artificielle** du jeu est répartie entre plusieurs composants autonomes : les contrats, les personnages, les groupes et organisations, et les chefs d'état en mode compétition.

Le **gestionnaire des contrats** s'occupe de la négociation des contrats économiques et traités diplomatiques par les chefs d'état *NPC*. Ces contrats sont soit proposés spontanément par les chefs d'état, soit par un joueur.

Un composant s'occupe également de la **réaction des personnages**. En effet, au sein d'une nation, nous retrouvons un ensemble de personnalités importantes, des politiciens aux chefs de syndicat, en passant par les artistes et autres sportifs.

Tous ces personnages réagissent aux actions (ou à l'inaction) d'un chef d'état, ce qui altère leur attitude face à ce dernier. Notons également que le peuple en général est considéré comme un personnage, et son attitude envers le chef d'état définit sa cote de popularité.

Les **groupes et organisations** sont gérés par un autre composant, qui se base cependant sur l'attitude des personnages envers le chef d'état. C'est ce composant qui va engager des actions de la part des différents groupes et organisations présents dans le jeu.

Ainsi, un syndicat mécontent lancera des grèves, une association environnementale engagera des manifestations pour dénoncer une politique qui ne respecte pas assez l'environnement, jusqu'à l'Union Européenne qui tapera sur les doigts des membres qui ne respectent pas le pacte de stabilité.

Enfin, en **mode compétition**, 16 pays s'affrontent pour devenir le pays le plus important au niveau mondial. Plusieurs joueurs peuvent ainsi prendre part au même jeu, et les pays qui ne sont pas choisis par un joueur sont gérés par une intelligence artificielle.

Cette IA, de manière semblable à notre solution, va décider d'actions à engager à chaque tour de jeu. Ces actions dépendent de la personnalité de l'IA, attribuée aléatoirement afin d'assurer une entropie satisfaisante. Ces actions ont pour but d'améliorer le score de l'IA, voire de saborder celui des *leaders*.

Wargame

Rulers of Nations comporte également un jeu de guerre (*Wargame*). En effet, il est possible pour des nations de s'affronter militairement, pour tenter d'affaiblir ou d'annexer un voisin gênant ou disposant de ressources intéressantes. Le chef d'état peut donc agir comme un chef de guerre et diriger ses armées au combat.

Afin de gérer cet aspect du jeu, plusieurs composants ont été développés. Le premier est le gestionnaire des unités, qui va s'occuper de l'intelligence des armées visibles sur la carte. Ainsi, il va se charger de calculer leur chemin pour qu'elles se rendent à leur destination, ainsi que les dégâts qu'elles subissent et infligent,

Le second composant est le **gestionnaire des conflits.** Comme son nom l'indique, il s'occupe de gérer le déroulement des conflits armés entre les nations : il organise la défense et l'attaque des *NPC* lorsqu'un conflit éclate et se poursuit, et implémente les conséquences d'un conflit qui se termine, à savoir les variations dans les territoires, et les conséquences économiques et politiques en cas d'annexion ou de colonisation.

Enfin, le **gestionnaire des relations diplomatiques** s'occupe de l'attitude des états entre eux. Ainsi, il se charge de définir l'attitude d'un état par rapport à un autre, d'honorer une alliance lors de l'éclatement d'un conflit, de faire respecter les droits et interdictions de passage et de survol des armées sur un territoire donné, et de déclencher un conflit en cas de fortes tensions.

Moteur économique

Le moteur économique est le composant qui s'occupe de gérer les aspects ô complexes de l'économie nationale et mondiale. Les informations économiques sont calculées et mises à jour à chaque *tour économique*. Par défaut, un tour économique se produit tous les 7 jours. Le moteur économique dispose de plusieurs sous-composants pour l'aider dans cette tâche.

Le premier est le **gestionnaire des secteurs d'activité**. En effet, chaque pays dispose de plusieurs centaines de secteurs, répartis entre l'agriculture, l'industrie, l'énergie et les services. Un secteur modélise l'ensemble des entreprises actives dans un domaine particulier, comme le blé, l'acier, les télécommunications, ou l'énergie fossile.

Chaque secteur dispose d'une production et de besoins en matériaux pour honorer cette production. A chaque tour, ce gestionnaire va donc calculer la production, les recettes et les dépenses de chaque secteur, et ce pour chaque nation. De plus, il va se charger de fermer un secteur si celui-ci venait à faire faillite.

La santé des différents secteurs va définir la croissance économique du pays, à savoir le développement du Produit Intérieur Brut (*PIB*).

Le **gestionnaire des devises** va, quant à lui, calculer la fluctuation des devises au niveau mondial. A chaque tour, la valeur d'une devise évolue, en fonction de plusieurs paramètres tels que le taux d'intérêt directeur et l'inflation des pays qui l'utilisent. De plus, ce gestionnaire s'occupe de convertir des montants d'une devise à une autre, à des fins de transferts ou de présentation au joueur.

Une devise intéressante est en effet le *dollar moteur*. Cette devise possède une valeur fixe, et est utilisée de manière interne au moteur afin que celui-ci manipule les montants économiques de manière homogène. Un montant présenté au joueur ou entré par le joueur subit donc une conversion en sortant ou en entrant dans le moteur économique.

Le **gestionnaire des budgets** va se charger de calculer les différents budgets de chaque état, et des indicateurs qui en découlent, comme le déficit budgétaire. Il se charge donc de récupérer les recettes et les charges de chaque ministère, et de calculer ainsi la balance d'une nation.

Enfin, afin de fournir au joueur des informations de référence, le **gestionnaire des historiques** va enregistrer les valeurs de plusieurs indicateurs au premier jour de chaque mois, et les présenter sous forme de graphiques. Il permet également de comparer ces indicateurs entre pays différents.

Réseau

Pour participer au mode *Compétition*, les joueurs peuvent créer des parties en **réseau**, que ce soit en local ou par Internet. Un composant se charge de gérer tout cet aspect réseau.

Une partie en réseau se déroule de manière centralisée : un serveur est créé par un joueur, qui héberge alors les autres joueurs sous forme de clients. Le serveur est le seul à disposer de l'ensemble des informations sur la partie en cours, et les clients ne sont que des présentations de ces informations récupérées du serveur.

Un client doit donc dialoguer constamment avec le serveur pour récupérer les informations nécessaires, et pour relayer les actions du joueur qu'il représente. Cela est géré par deux sous-composants : le gestionnaire des connexions, et le gestionnaire des échanges de données.

Le **gestionnaire des connexions** s'occupe de mettre en relation les serveurs et les clients au lancement d'une partie, mais également au cours d'une partie. Il établit ainsi le « pont » entre deux instances du jeu, leur permettant de communiquer actions et informations.

Le **gestionnaire des échanges de données** récupère les demandes d'informations de la part d'un client, les envoie au serveur qui va rassembler les informations demandées, et récupère ensuite la réponse.

Fichiers

Un grand nombre de structures de données sont définies afin de pouvoir aux besoins d'information du jeu. En effet, la simulation a besoin d'un grand nombre de données à tout moment afin de s'exécuter : la multitude d'informations sur les nations, les personnages, les groupes et organisations, les actions, etc.

Un composant se charge donc de définir, garnir et sauvegarder toutes ces données, à l'aide de **fichiers** externes. Ce composant comporte entre autres un sous-composant qui va s'occuper de définir toutes les structures de données qui vont accueillir les différentes informations nécessaires.

Il comporte également un **gestionnaire des profils et sauvegardes**. En effet, un joueur participe au jeu sous couvert d'un profil de joueur, qui reprend un nom, des points et un historique de parties jouées.

Un joueur peut également sauvegarder une partie en cours. Ce processus de sauvegarde nécessite d'écrire toutes les informations dynamiques du jeu dans un fichier de sauvegarde, qui pourra alors être lu à un moment ultérieur afin de reconstruire la simulation dans l'état dans laquelle elle se trouvait au moment de la sauvegarde.

Ces informations sont contenues dans les différentes structures de données du jeu. Ce composant va donc se charger d'écrire et de lire ces différentes structures dans un fichier de sauvegarde, mais également d'assurer la compatibilité entre des sauvegardes de versions différentes. En effet, les structures de données peuvent changer entre les versions du jeu ; il faut donc s'assurer que les sauvegardes antérieures restent compatibles avec la version courante du jeu.

Enfin, un **gestionnaire des bases externes** est présent afin de se charger du dialogue avec les bases externes. Ce gestionnaire va s'occuper de « compiler » les bases en une source de données utilisable directement par le jeu, et s'assurera de l'interprétation de ces données au cours de la simulation.

Bases externes

La simulation se base sur un ensemble de données tirées de la réalité, comme les nations et leurs différents indicateurs, les villes, les personnages, etc. Toutes ces données sont rassemblées dans des **bases externes**, qui sont construites comme des tableurs structurés.

Nous retrouvons parmi ces bases, une **base des actions** qui reprend la définition de toutes les actions à disposition d'un joueur. Ces actions reprennent donc les modifications de budget, la modification de taxes, la subvention de secteurs d'activité, etc.

Une **base des réactions** est également présente. Elle décrit l'ensemble des événements et réactions qui peuvent se produire au cours de la simulation. Ces événements sont regroupés en « *pools* » identifiés par un code, et chaque événement dispose également d'un indice.

Les événements peuvent donc être déclenchés au cours de la simulation en s'y référant par leur code de *pool*, ou leur indice. Un indice référencera une réaction directement, tandis que le *pool* permettra de choisir cette réaction sur base des différents paramètres qui le compose.

Une **base des nations** reprend les différentes informations sur les nations du jeu – leur population, leur déficit, leur dette, leur PIB, etc. De la même manière, une **base des personnages** reprend la définition de toutes les personnalités, les groupes et les associations du jeu.

La **base IA** reprend quant à elle les différentes actions qu'une IA peut entreprendre au cours du mode *Compétition*, ainsi que leurs différents paramètres. Ces actions sont regroupées en catégories, afin de permettre leur différenciation par les personnalités des contrôleurs.

On retrouve également une base comprenant tous les **textes** du jeu, dans les différentes langues supportées. Ces textes sont regroupés en *pools*, et sont référencés par leur *pool* et leur position dans ce *pool*.

La **base des activités** reprend la définition de tous les secteurs d'activités du jeu. Ces secteurs sont présentés sous forme de catégories et disposent de plusieurs paramètres, comme l'unité de ce qu'ils produisent (Tonnes, Twh, …). Ces secteurs sont référencés par un code identifiant.

Une autre base majeure est la **base des organisations**. Cette base reprend la définition des différentes organisations internationales du jeu, leurs membres et leurs intérêts.

Il existe de nombreuses autres bases externes mineures, qui contiennent par exemple les noms et emplacements des différents lieux géographiques, la définition des villes, ou encore la définition des différents scénarios jouables en simulation.

Librairie de base

Notons également que tous les composants décrits intègrent des outils provenant d'une librairie de base offrant un ensemble de structures de données, de fonctions et de procédures utiles à l'implémentation du jeu. Cette librairie se nomme la *Geol* et est propriétaire au studio de développement *Eversim*.

Cette librairie redéfinit donc par exemple tous les types primitifs et les structures de base, comme les tableaux, ceci afin de mieux contrôler l'espace mémoire. Nous pouvons reconnaître les types et fonctions provenant de la *Geol* sur base de leur nom. En effet, par convention, tous les noms d'outils provenant de la librairie comment par un « _ ». Ainsi par exemple, le type entier est noté « _*int* ».

4.1.4 Structures de données

La simulation nécessite un grand nombre de **données** pour pouvoir s'exécuter. La majeure partie de ces données est gardée en mémoire sous la forme de structures de données. Nous présentons ici les structures les plus importantes.

La structure la plus importante est sans aucun doute la structure *Nation*. Cette structure reprend toutes les informations pertinentes pour une nation, comme ses différents budgets, ses indicateurs économiques, son chef d'état, son statut de *PC* ou *NPC*, les groupes qu'elle héberge, ses indicateurs géographiques et sociaux, etc.

Les groupes et les personnalités d'une nation sont regroupés dans deux structures : la structure *Groupe* et la structure *Perso*. La structure *Groupe* reprend les différentes informations à propos d'un groupe : son nom, son chef, son logo, ses intérêts, son importance, etc.

La structure *Perso* reprend la définition des différentes personnalités d'une nation : son nom, son sexe, son âge, son portrait, sa fonction, son attitude vis-à-vis de son chef d'état, ses intérêts, etc.

De la même manière, les organisations internationales sont définies au travers de la structure *Org*. On y retrouve leurs membres, leurs intérêts, leur président, l'agenda de leurs sommets, etc.

Chaque nation comporte nombre de secteurs d'activité, qui dépendent de deux structures. La première, *Defactivite*, reprend la définition de chaque activité, tandis que la deuxième, *Activite*, instancie chaque activité auprès des différentes nations.

Defactivite comprend ainsi le type de l'activité, ce qu'elle produit, ce dont elle a besoin, etc. *Activite* reprend, quant à elle, la production courante de l'activité, son prix, le personnel qu'elle occupe, son chiffre d'affaires, etc.

Toutes les actions qu'un chef d'état peut entreprendre sont définies dans une structure *Defaction*. De la même manière, chaque réaction et événement est définit dans une structure *Reaction*. Ces structures reprennent principalement le type de l'action ou de la réaction et leurs paramètres.

Tout comme les bases externes, il existe encore de nombreuses autres structures de données, comme par exemple celles qui encapsulent les échanges de données lors de parties en réseau. Les structures présentées ici sont cependant les plus importantes.

4.2 Implémentation

Maintenant que le contexte du produit est planté, nous pouvons aborder l'implémentation de la solution au sein de ce produit. Tout d'abord, nous allons décrire les choix qui guideront notre implémentation. Par après, nous décrirons notre définition des valeurs des constantes et des structures de données. Ensuite, nous reprendrons chaque spécification définie au cours de la recherche des solutions, et présenterons les choix et spécificités qui définiront leur implémentation.

4.2.1 Choix d'implémentation

Au sein de notre implémentation, nous n'allons travailler qu'avec les variables d'environnement qui nous semblent les plus critiques : **la popularité, l'alignement politique, et le déficit.**

De fait, le PIB et l'inflation sont fortement liés à la croissance – influer sur cette dernière revient à influer sur ces variables. Et la croissance influe elle-même sur la proportion du déficit.

En effet, si elle n'agit pas directement sur la valeur même du déficit, l'importance de celui-ci est calculé en faisant le rapport avec le PIB (on parle ainsi de *déficit en % du PIB*). La dette de l'état est aussi liée au déficit budgétaire – réduire le déficit impliquera une réduction de la dette potentielle.

Nous reprenons par contre **toutes les actions disponibles** : augmenter ou diminuer le budget, l'effectif ou les salaires d'un ministère ; augmenter/créer ou diminuer un taxe ; subventionner un secteur d'activité ; assister à un événement populaire ; visiter un lieu public ; et donner une allocution à la télévision.

Nous ajoutons également une **dernière action**, qui consiste à **ne rien faire** (*NOP*). Il se peut effectivement que ne rien faire soit plus profitable que d'agir.

L'ensemble des actions considérées sera repris dans un **fichier de définition externe**. Ce fichier sera donc structuré comme un tableur, avec la première colonne correspondant aux indices des actions considérées, et les colonnes suivantes étant disponibles pour tout ajout ultérieur de paramètres.

Ce fichier présente également l'avantage qu'un membre de l'équipe de développement qui n'est pas technicien peut mettre à jour les actions à prendre en compte. Ainsi par exemple, un *game designer* peut modifier ce fichier pour qu'il corresponde à sa conception du jeu.

4.2.2 Constantes et structures de données

Nous allons décrire nos choix en termes de valeur pour les constantes et de structure pour les variables, d'abord pour la modélisation générale, ensuite pour les deux solutions proposées.

Général

Le nombre d'indicateurs, *n*, est de 3 : le déficit, la popularité et l'alignement politique du chef d'état. Le nombre d'actions, *m*, dépend des informations contenues dans le fichier externe de définition.

Le nombre d'actions à choisir, *s*, est fixé à 3. Ces actions s'exécuteront tous les 7 jours, comme le *tour économique*. Afin de ne pas surcharger ce tour de calculs, nous exécuterons ces actions 3 jours après chaque tour économique.

Les structures de données à définir sont une liste (*Choices* et *V*) et une matrice (*M*). Nous choisissons de les représenter comme des tableaux à 1 et 2 dimensions respectivement, vu que leur taille est fixe (*s* pour *Choices*, *n* pour *V*, et $m * (n + 1)$ pour *M*).

Recherche complète

Les constantes de la recherche par le tabou demandent un travail d'ajustement empirique. Nous partons de ces valeurs arbitraires que nous ajusterons en fonction des résultats : 30 solutions de base générées, une taille de voisinage de 10 éléments, 50 itérations, et une taille de liste taboue de 20 éléments.

Pour ce qui est du seuil d'aspiration, nous choisissons de ne pas l'exprimer et le calculer en absolu, mais plutôt en relatif. En effet, le domaine de la fonction objectif étant très grand, il nous est difficile d'estimer un seuil constant.

Nous décidons donc de ne pas comparer la différence entre les deux valeurs de la fonction-objectif, mais plutôt leur rapport. Le seuil d'aspiration serait alors posé à 15%.

Tout comme les autres listes, nous choisissons d'utiliser des tableaux pour représenter *ListeSolution* et la liste des tabous. *ListeSolution* sera un tableau à une dimension de taille fixe, 30 pour les solutions de base et 10 pour le voisinage.

La liste des tabous sera quant à elle, un tableau à deux dimensions. La première dimension correspond à l'indice de l'action, et la deuxième à l'emplacement dans la solution. Leur taille serait identique : 20. Ainsi, les couples tabous sont représentés par la conjonction de deux cellules de même indice dans les deux dimensions du tableau.

Décomposition en sous-recherches

Pour la deuxième solution, nous n'avons pas de constantes à spécifier, mais simplement une structure de données à définir : celle de *PoidsActions*. Comme pour les autres listes, nous choisissons de la représenter par un tableau, de taille *m*.

4.2.3 Spécification 1

Nom : - Solution au problème d'optimisation géopolitique
Input : - La liste des actions considérées à disposition d'un chef d'état. - La liste des valeurs des variables d'environnement du problème. - La nation à laquelle correspondent les variables d'environnement. - Les pondérations correspondant à l'importance accordée par la nation considérée à chaque variable d'environnement.
Output : - Une séquence de *s* numéros d'action, identifiant des actions à exécuter parmi les *m* actions disponibles au contrôleur.
Dépendances : - Une fonction d'utilité pour chacune des *n* variables d'environnement considérées par la solution. - Une fonction-objectif donnant une mesure à partir des valeurs d'utilités des variables d'environnement.

Spécification 1 : Solution au problème d'optimisation géopolitique

Cette spécification correspond au pseudo-code des deux solutions que nous avons proposées dans le chapitre 2. Ce pseudo-code étant déjà suffisamment bien décrit que pour recevoir une implémentation directe, nous devons cependant nous intéresser à commet nous allons récupérer les *inputs*, et comment nous allons exécuter les *outputs*.

Inputs

La liste des actions considérées sont répertoriées dans un fichier externe, qui constituera une nouvelle *base externe* (voir 4.1.3, paragraphe *Bases externes*). Afin de rester cohérent avec le traitement des autres bases, ce fichier sera lu au lancement du jeu et la liste des actions sera créée à ce moment. La constante *m* sera donc instanciée après lecture de cette base, et la liste des actions sera construite comme un tableau statique de taille *m*.

Les variables d'environnement à fournir en entrée sont au nombre de 3 : la popularité du chef d'état, le déficit de la nation et l'alignement du chef d'état par rapport à son parti. La popularité est récupérée dans la structure *Perso* du chef d'état. Le déficit est récupéré dans la structure *Nation* de la nation.

Quant à l'alignement du chef d'état avec son parti, nous faisons appel à une fonction préexistante, *alignement_president(Perso p, Nation n)*, qui renvoie cette information sur base de la structure *Perso* du chef du parti du président et de la structure *Nation* de sa nation.

Les pondérations des variables d'environnement sont pour le moment identiques pour chaque nation, à savoir 5 pour le déficit, 3 pour la popularité, et 2 pour l'alignement. Ces pondérations sont arbitraires et correspondent à l'ordre d'importance des indicateurs dans le développement de notre solution.

La récupération de toutes ces informations et l'appel à une solution sur toutes les nations se fera dans une fonction codée à cet effet. Cette fonction bouclera sur toutes les nations afin d'exécuter la solution.

Outputs

Une fois la liste des actions à exécuter générée, il nous reste cependant à mener ces actions à bien pour toutes les nations. Nous allons donc faire appel à une fonction qui s'occupera d'exécuter chaque action que nous lui renseignerons.

Cette fonction effectuera un prétraitement si nécessaire en fonction de l'action. Par exemple, si l'action correspond à augmenter une taxe, il est peut-être nécessaire de la créer, si elle n'existe pas encore dans le pays en question. Ensuite, cette fonction enverra l'action et ses paramètres au gestionnaire des actions du moteur principal, à l'aide de la fonction préexistante *send_action*.

4.2.4 Spécification 2

Nom : - Fonction d'utilité d'une variable d'environnement
Input : - Une variable d'environnement - La nation à laquelle correspond la variable d'environnement
Output : - Une valeur d'utilité mesurant la valeur de la variable d'environnement pour la nation considérée
Dépendances : - Aucune.

Spécification 2 : Fonction d'utilité d'une variable d'environnement

Il est nécessaire ici de choisir quelle fonction d'utilité nous allons associer aux variables d'environnement. Tout comme les pondérations des indicateurs, ces fonctions d'utilité sont identiques pour toutes les nations. Les variables à évaluer sont au nombre de 3 : le déficit, la popularité et l'alignement politique.

Pour le déficit, nous nous basons sur l'évaluation décrite dans la section 3.1.3, qui demande qu'il soit au maximum égal à 3% du PIB. Nous choisissons donc de composer deux fonctions : avant ces 3%, l'utilité du déficit croît exponentiellement – ainsi, plus le déficit est important, au pire l'indicateur sera. A partir de 3%, l'utilité croît linéairement.

Pour la popularité, nous décidons de définir plusieurs « zones » : critique, faible, correcte, et élevée. Chaque zone indique un degré d'importance à accorder à la popularité. Nous définissons ces zones entre bornes de pourcentages comme suit :

$$Critique = [0,20[\ ; Faible = [20,50[\ ; Correcte = [50,70[\ ; Elevée = [70,100]$$

A chaque zone, nous associons une fonction d'utilité linéaire croissante, dont la pente est moins importante au plus on grimpe dans les zones. Ainsi, gagner 1% de popularité dans la zone critique sera beaucoup plus intéressant que dans la zone élevée.

Enfin, pour l'alignement politique, nous reprenons directement la valeur donnée en *input* par la fonction *alignement_president*, vu que celle-ci communique déjà une valeur entre -100 et 100 correspondant à une telle utilité.

4.2.5 Spécification 3

Nom : - Fonction-objectif du problème d'optimisation géopolitique
Input : - La valeur d'utilité des *n* variables d'environnement considérées - La pondération de la valeur d'utilité de chacune des *n* variables d'environnement considérées
Output : - Une mesure correspondant à la somme des utilités pondérées.
Dépendances : - Aucune.

Spécification 3 : Fonction-objectif du problème d'optimisation géopolitique

Cette implémentation est directe et résulte de (3.3).

4.2.6 Spécification 4

Nom : - Génération des solutions de base de l'algorithme
Input : - Le nombre de solutions à générer.
Output : - Une liste de solutions de taille correspondante au nombre donné en entrée. Une solution correspond à une instance de *Choices*.
Dépendances : - Un générateur aléatoire uniforme. (Spécification 13)

Spécification 4 : Génération des solutions de base de l'algorithme

Il s'agit d'utiliser le générateur aléatoire pour générer le nombre de solutions requises, en garnissant les *s* cellules des éléments du tableau à renvoyer de nombres aléatoires entre 1 et *m*.

4.2.7 Spécification 5

Nom : - Choix de la meilleure solution parmi une liste de solutions
Input : - Une liste de solutions.
Output : - Une solution qui correspond à la meilleure solution vis-à-vis de l'impact sur la fonction-objectif.
Dépendances : - Une fonction qui calcule l'impact d'une solution sur la fonction-objectif. (Spécification 12)

Spécification 5 : Choix de la meilleure solution parmi une liste de solutions

Il s'agit de boucler sur la liste de solutions en appliquant la Spécification 12 et en retenant la meilleure solution, qu'il s'agira de renvoyer.

4.2.8 Spécification 6

Nom : - Génération du voisinage d'une solution courante
Input : - Une solution correspondant à la solution courante. - La taille du voisinage à générer.
Output : - Une liste de solutions qui correspond au voisinage de la solution courante renseignée, et dont la taille équivaut au nombre renseigné.
Dépendances : - Une fonction qui génère une nouvelle solution à partir d'une solution courante. (Spécification 14)

Spécification 6 : Génération du voisinage d'une solution courante

Il s'agit de boucler un nombre de fois égal à la taille du voisinage en appliquant la Spécification 14 sur la solution courante, et en s'assurant que la solution générée n'existe pas déjà dans le voisinage.

4.2.9 Spécification 7

Nom : - Vérification de l'interdiction d'une solution
Input : - Une solution correspondant à la solution considérée. - La liste des couples tabous.
Output : - Vrai si la solution renseignée est interdite, Faux sinon.
Dépendances : - Aucune.

Spécification 7 : Vérification de l'interdiction d'une solution

Il s'agit de boucler sur la liste des couples tabous et de vérifier qu'à l'emplacement indiqué de la solution, l'action taboue n'est pas présente.

4.2.10 Spécification 8

Nom : - Vérification du critère d'aspiration d'une solution.
Input : - Une solution correspondant à la solution considérée. - La meilleure solution actuellement retenue.
Output : - Vrai si la solution considérée vérifie le critère d'aspiration et donc ne peut être interdite ; Faux sinon.
Dépendances : - Une fonction qui calcule l'impact d'une solution sur la fonction-objectif. (Spécification 12)

Spécification 8 : Vérification du critère d'aspiration d'une solution

Le critère d'aspiration étant exprimé comme un pourcentage par choix d'implémentation, il s'agit de faire le rapport entre la valeur de la fonction-objectif pour la solution taboue et la valeur pour la meilleure solution, et de tester si ce rapport égale ou dépasse le critère.

4.2.11 Spécification 9

Nom : - Suppression d'une solution d'une liste de solutions.
Input : - La liste de solutions. - La solution à enlever de la liste de solutions.
Output : - Aucun.
Dépendances : - Aucune.

Spécification 9 : Suppression d'une solution d'une liste de solutions.

Il s'agit de boucler sur la liste de solutions et de retirer la première solution qui correspond à celle renseignée en entrée. La liste étant un tableau cependant, il faudra décaler toutes les solutions suivant la solution retirée afin de ne pas laisser de « trou » dans la séquence.

4.2.12 Spécification 10

Nom : - Calcul d'un nouveau couple tabou
Input : - Une solution correspondant à la solution courante. - Une solution correspondante à la nouvelle solution considérée et qui ne diffère que d'un élément de la solution courante.
Output : - Le couple tabou qui correspond à l'élément de la solution courante qui a été modifié et de son emplacement dans la solution.
Dépendances : - Aucune.

Spécification 10 : Calcul d'un nouveau couple tabou

Il s'agit de boucler sur les deux solutions en même temps, et de comparer les indices des actions aux mêmes emplacements. Une fois que les actions seront différentes, on retient l'emplacement et l'action de la solution courante, ce qui constitue le couple tabou.

4.2.13 Spécification 11

Nom : - Ajout d'un nouveau couple tabou à la liste des tabous
Input : - La liste des tabous. - Le couple tabou à ajouter à la liste des tabous.
Output : - Aucun.
Dépendances : - Aucune.

Spécification 11 : Ajout d'un nouveau couple tabou à la liste des tabous

Il s'agit simplement de décaler tous les éléments courants du tableau représentant la liste des tabous d'un emplacement, et d'éventuellement supprimer celui qui dépasserait de la taille du tableau, pour ensuite ajouter le couple dans la première cellule.

4.2.14 Spécification 12

Nom : - Calcul de l'impact d'une solution sur l'objectif
Input : - Une solution correspondant à la solution considérée.
Output : - La différence entre la valeur estimée de la fonction-objectif après exécution de l'action et la valeur courante.
Dépendances : - Une fonction d'utilité par variable d'environnement considérée par la fonction-objectif. (Spécification 2) - Une fonction calculant la valeur de la fonction-objectif. (Spécification 3) - Une fonction calculant l'impact d'une action sur les variables d'environnement. (Spécification 15 : Calcul de l'impact d'une action sur l'environnement

Spécification 12 : Calcul de l'impact d'une solution sur l'objectif

Il s'agit de boucler sur la solution renseignée en appliquant la Spécification 15 : Calcul de l'impact d'une action sur l'environnement sur chacun de ses éléments, puis d'appliquer la Spécification 2 sur chaque variable d'environnement ainsi mise à jour pour en tirer les indicateurs, afin d'appeler la Spécification 3 pour calculer la valeur estimée de la fonction-objectif pour la solution. Il suffira alors de faire la différence entre la valeur calculée de la fonction-objectif et la valeur courante.

4.2.15 Spécification 13

Nom : - Génération d'une valeur aléatoire uniforme
Input : - La borne supérieure pour la valeur aléatoire.
Output : - Une valeur réelle aléatoire entre 0 et la borne supérieure spécifiée. La génération doit être uniforme, c'est-à-dire que chaque valeur a la même probabilité d'être tirée.
Dépendances : - Aucune.

Spécification 13 : Génération d'une valeur aléatoire uniforme

Un générateur aléatoire existe déjà dans le produit, et peut être invoqué à l'aide de la fonction *_randf(_flt min, _flt max)*. L'implémentation de cette fonction revient à un appel à cette fonction en renseignant 0 pour *min* et la borne supérieure pour *max*.

4.2.16 Spécification 14

Nom : - Génération d'un voisin d'une solution courante.
Input : - La solution courante.
Output : - Une nouvelle solution qui ne diffère que d'un élément de la solution courante.
Dépendances : - Un générateur aléatoire uniforme. (Spécification 13)

Spécification 14 : Génération d'un voisin d'une solution courante.

Il s'agit d'utiliser le générateur aléatoire pour produire deux nombres. Le premier, entre 1 et *s*, correspondra à l'indice de l'emplacement de l'action à changer dans la séquence. Le deuxième, entre 1 et *m*, correspondra à l'indice de l'action qui remplacera l'actuelle. Il faudra cependant vérifier que les deux actions ne sont pas identiques, auquel cas il faudra tirer une nouvelle action.

4.2.17 Spécification 15

Nom : - Calcul de l'impact d'une action sur l'environnement
Input : - Le numéro de l'action choisie. - La liste des variables d'environnement considérées.
Output : - La liste des variables d'environnement est mise à jour avec l'impact de l'action choisie.
Dépendances : - Aucune.

Spécification 15 : Calcul de l'impact d'une action sur l'environnement

Il s'agit ici d'estimer l'impact d'une action sur les différentes variables d'environnement. Pour rappel, les variables d'environnement considérées sont la popularité, le déficit et l'alignement avec le parti.

Pour la popularité, il existe une fonction nommée *cumul_orient_action* qui va appliquer l'impact d'une action sur les différentes personnalités. Nous rajoutons simplement un mode d'estimation qui va calculer la différence entre la valeur courante et la valeur après application de l'action, en nous gardant toutefois de sauvegarder l'impact. Il restera ensuite à appeler cette fonction sur le peuple en mode estimation en renseignant l'action.

Pour l'alignement politique, nous agissons de la même manière que pour la popularité, en rajoutant un mode estimation à la fonction *alignement_president*. Enfin, pour le déficit, il existe un mode pour la fonction *send_action* qui calcule l'impact d'une action sur le budget de l'état. Il suffit donc de récupérer ce résultat et de l'appliquer sur le déficit.

4.2.18 Spécification 16

Nom : - Evaluation de l'impact des actions considérées
Input : - La liste des actions et de leur poids qui sera garnie par la fonction. - La variable devant contenir la somme des poids positifs et qui sera garnie par la fonction.
Output : - La liste des actions et de leur poids a été garnie. - La variable devant contenir la somme des poids a été garnie.
Dépendances : - Une fonction d'utilité par variable d'environnement considérée par la fonction-objectif. (Spécification 2) - Une fonction calculant la valeur de la fonction-objectif. (Spécification 3) - Une fonction calculant l'impact d'une action sur les variables d'environnement. (Spécification 15 : Calcul de l'impact d'une action sur l'environnement

Spécification 16 : Evaluation de l'impact des actions considérées

Il s'agit ici de boucler sur la liste des actions en appliquant la Spécification 15 sur chaque action, pour ensuite retirer les indicateurs des variables d'environnement à l'aide de la Spécification 2, pour enfin calculer la valeur de la fonction-objectif par la Spécification 3.

Il s'agira ensuite de faire la différence entre la valeur ainsi obtenue et la valeur courante de la fonction-objectif pour obtenir le poids d'une action, et l'ajouter à la somme des poids si celui-ci est positif.

4.2.19 Spécification 17

Nom : - Tri de la liste des actions et de leur poids
Input : - La liste des actions et de leur poids.
Output : - La liste des actions et de leur poids a été triée par ordre décroissant de poids.
Dépendances : - Aucune.

Spécification 17 : Tri de la liste des actions et de leur poids

Cette implémentation est un simple tri par ordre décroissant. Nous choisissons d'implémenter l'algorithme du *Quicksort* pour trier ce tableau, sur base des valeurs des poids.

4.2.20 Spécification 18

Nom : - Génération de l'arbitre
Input : - La somme des poids positifs des actions considérées.
Output : - Une valeur générée aléatoirement entre 0 et la somme des poids renseignée.
Dépendances : - Un générateur aléatoire uniforme. (Spécification 13)

Spécification 18 : Génération de l'arbitre

Il s'agit ici simplement d'un appel au générateur aléatoire (Spécification 13) avec comme borne supérieure la somme des poids.

4.3 Conclusion

Au cours de ce chapitre, nous avons présenté l'environnement dans lequel le logiciel-cible de l'implémentation, *Rulers of Nations*, évoluait. Nous avons ainsi décrit le studio qui a développé ce produit, ainsi que ses principes d'utilisation majeurs.

Ensuite, nous avons décrit l'architecture logique du logiciel, en présentant tous les modules majeurs qui le composent. Nous avons également décrit les différentes structures de donnée que le produit utilise.

Par après, nous avons décrit nos choix d'implémentation pour la valeur des constantes et la structure des variables utilisées, avant de décrire comment nous allions implémenter chacune des spécifications découvertes par la modélisation.

L'étape suivante est de mener l'implémentation de ces solutions et d'ensuite en conduire l'évaluation. Cette évaluation fera l'objet du chapitre suivant.

Chapitre 5 - Etude de cas : évaluation de la solution

Nous avons maintenant implémenté les propositions de solution, et nous devons conduire leur évaluation – étudier leur accord avec les objectifs posés initialement, mettre en évidence leurs points forts et leurs points faibles, et étudier les améliorations que nous pourrions leur porter.

Ensuite, nous reviendrons sur le processus de recherche qui nous a amené à définir ces solutions, et nous poserons un regard critique sur notre démarche, en soulignant ses aspects positifs et négatifs, et en suggérant des améliorations.

5.1 Evaluation des solutions

Pour évaluer les solutions, nous nous basons sur les critères définis dans la section 3.1.3. Pour rappel, ces critères concernaient le déficit, l'entropie et l'évolutivité.

Pour satisfaire au critère de déficit, une solution doit faire en sorte qu'au 1er janvier 2050, 90% des pays aient un déficit inférieur ou égal à 3% de leur PIB. Par soucis de comparaison, sans l'application des solutions, au 1er janvier 2050 23% des pays seulement ont un déficit inférieur ou égal à 3% de leur PIB.

Pour satisfaire au critère d'entropie, les décisions prises par les différents pays doivent être différentes entre chaque nouvelle partie. Enfin, pour satisfaire au critère d'évolutivité, l'ajout de nouveaux indicateurs ou actions ne doit nécessiter aucune restructuration des solutions.

5.1.1 Exploration complète

Evaluation

Au cours de 5 parties différentes, la solution d'exploration complète a toujours produit comme résultat une proportion de pays supérieure à 90% ayant leur déficit supérieur ou égal à 3% de leur PIB. La moyenne fut de 92%.

Concernant l'entropie, chaque décision est prise sur base d'un ensemble aléatoire de solutions. Même si la meilleure solution est toujours choisie, les décisions sont donc généralement forts différentes.

Enfin, la solution est évolutive. En effet, il suffit d'ajouter des actions dans le fichier externe de définition et dans la fonction d'exécution de ces actions pour que nouvelles actions soient prises en compte.

De plus, il suffit de rajouter les fonctions d'utilité nécessaires et d'étendre la fonction-objectif s'il devenait nécessaire de rajouter des variables d'environnement à considérer.

La solution de l'exploration complète satisfait donc aux objectifs posés.

Points forts

L'exploration complète possède plusieurs grands avantages : l'évolutivité, la flexibilité, et l'entropie. En effet, la solution parcourt les solutions possibles de manière aléatoire et dirigée. De ce fait, il ne lui est pas nécessaire de parcourir l'ensemble des actions à la définition de chaque nouvelle solution.

Cet avantage lui permet donc de fonctionner équitablement bien sur un nombre d'actions qui peut être réduit ou très grand. Sa précision sera d'autant plus grande que le nombre d'actions est petit, mais ceci peut être ajusté en modifiant ses paramètres – par exemple, la taille du voisinage et le nombre d'itérations.

Cette flexibilité est également intéressante du point de vue de l'équilibre entre le temps de calcul et la précision requise. En effet, si le temps qu'occupe l'exploration pour définir des solutions est trop grand, la précision peut être diminuée pour gagner du temps de calcul – et l'inverse est également possible.

Enfin, cette solution présente une entropie importante – vu que les solutions sont générées aléatoirement, elles ont de grandes chances d'être différentes d'une partie à l'autre. Cette entropie est cependant d'autant plus faible que la précision grandit – à haute précision, il deviendrait peut-être nécessaire d'utiliser un système d'arbitre comme pour la deuxième solution.

Points faibles

Le plus gros point faible de cette solution est la paramétrisation – car en effet si sa flexibilité est une force pour l'exploration, cela demande également un temps non négligeable d'ajustement empirique afin d'obtenir le résultat escompté. L'impossibilité de prédire le résultat à l'avance est donc un défaut sensible.

Le second défaut est la complexité de cette solution. En effet, la complexité théorique en temps dans le pire des cas pour cette solution est de l'ordre de $O(n^4)$ (voir section 3.3.1 – Complexité). Ainsi, à haute précision elle demande un nombre non-négligeable de calculs, ce qui risque de poser problème quant au temps d'exécution.

Améliorations

Un des problèmes de cette solution est que l'on répète beaucoup de fois le même calcul – à savoir l'évaluation de l'impact d'une action. Cette évaluation ne devrait cependant pas donner de résultats sensiblement différents d'une itération à une autre pour la même action.

Une amélioration possible serait donc d'introduire un système de *cache*, où à chaque fois que l'impact d'une action sur les variables d'environnement est calculé, celui-ci est sauvegardé et peut être consulté à un moment ultérieur dans l'exécution de l'algorithme. Cela économiserait un nombre non-négligeable de calculs pour une faible perte de précision.

5.1.2 Décomposition en sous-recherches

Evaluation

Au cours de 5 parties différentes, la solution de décomposition en sous-recherches a toujours produit comme résultat une proportion de pays supérieure à 90% ayant leur déficit supérieur ou égal à 3% de leur PIB. La moyenne fut de 96%.

Concernant l'entropie, chaque décision est prise sur base d'une variable aléatoire, l'arbitre. Cette variable assure que les meilleures décisions soient le plus souvent choisies, mais permet également à des décisions moins bonnes d'être engagées, ce qui fait que les actions des *NPC* seront différentes d'une partie à une autre.

Enfin, la solution est évolutive. En effet, tout comme la première solution, il suffit d'ajouter des actions dans le fichier externe de définition et dans la fonction d'exécution de ces actions pour que nouvelles actions soient prises en compte.

De plus, il suffit de rajouter les fonctions d'utilité nécessaires et d'étendre la fonction-objectif s'il devenait nécessaire de rajouter des variables d'environnement à considérer.

La solution de décomposition en sous-recherches satisfait donc aux objectifs posés.

Points forts

Le plus gros point fort de la décomposition en sous-recherches est sans nul doute sa considération de l'ensemble des actions à chaque exécution. Cela permet ainsi de ne laisser aucune action de côté et d'établir un classement complet des décisions.

Le second avantage de cet algorithme est de diriger la recherche sur base des meilleures décisions. En effet, la meilleure action est toujours mise en avant dans la composition de chacun des éléments de la solution globale, ce qui a pour effet d'amener plus rapidement le pays dans le vert.

Points faibles

Le point faible le plus important de cette solution est sa rigidité. En effet, elle peut sans doute très bien faire face à un faible nombre d'actions, mais si jamais celui-ci devait être grand, la solution prendrait un temps considérable en plus à définir les solutions, vu qu'elle considère automatiquement toutes les actions.

Cette solution présente donc un défaut d'évolutivité, qui risque de l'handicaper grandement si le jeu évolue en proposant un nombre croissant d'actions.

Améliorations

Tout comme la première solution, un système de *caching* permettrait d'éviter un grand nombre de calculs. En effet, d'une itération à l'autre, l'impact de toutes les actions est évalué. Sauvegarder cet impact permettrait de ne le calculer qu'une seule fois par pays, ce qui consisterait en une optimisation non-négligeable de la solution.

5.1.3 Comparaison des solutions

Nous établissons ici un tableau comparatif des solutions sur base de leurs caractéristiques les plus importantes.

Caractéristique	Exploration complète	Décomposition en sous-recherches
Evaluation	10/10	10/10
Flexibilité	Forte	Faible
Entropie	Variable selon les paramètres	Moyenne
Paramétrisation	Forte	Faible
Temps de calcul	Variable selon les paramètres	Variable selon le nombre d'actions à considérer
Convergence	Variable selon les paramètres	Forte
Redondance des calculs	Forte	Forte

Nous mettons donc en évidence que l'exploration complète est la solution qui apporte sans doute le moins rapidement des résultats, mais est très évolutive au vu de sa flexibilité. Elle n'aura aucun mal à s'adapter à des problèmes de grande taille. A contrario, la décomposition en sous-recherches convient pour des problèmes de taille faible à modérée, et pour lesquels le temps de calcul alloué à la solution est grand.

Nous estimons donc que dans l'état actuel des choses, la décomposition en sous-recherches est sans doute la solution la plus utile, tandis que si le nombre d'actions à considérer ou le nombre de décisions à produire devait augmenter, l'exploration complète serait à considérer.

De plus, si l'énoncé du problème venait à changer – c'est-à-dire si le besoin de précision venait à évoluer – l'exploration complète serait également la meilleure réponse à notre avis.

5.2 Evaluation de la démarche

Nous voici à présent au terme de notre démarche de recherche d'une solution à un problème vidéo-ludique, en mettant en avant le processus de pré-production qui conduit à une modélisation et une décomposition poussée. Il est donc temps de poser un regard critique sur cette démarche, et d'en retirer les points forts et les points faibles.

5.2.1 Evaluation

Notre démarche de définition et de recherche de solution a demandé 22 heures de travail, pour une implémentation en demandant 16. La démarche a donc coûté au total 38 heures pour un travailleur.

Par rapport à notre expérience « sur le terrain », cela diffère de la démarche de production en prenant sans surprise plus de temps sur la recherche et la conception que sur l'implémentation.

Il est difficile de comparer avec d'autres expériences au vu de la différence fondamentale de type et de complexité des problèmes. Nous pouvons cependant affirmer qu'une tâche que nous avons dû mener à bien et qui comportait une taille et une complexité approximativement équivalente, nous avons passé plus de temps à modéliser la solution, mais beaucoup moins de temps à l'implémentation.

Cette tâche en question fut l'implémentation d'un système de privatisation et de nationalisation des secteurs d'activités au sein d'une nation. La démarche de modélisation et de documentation demanda 12 heures de travail, tandis que l'implémentation en demanda presque 60.

Nous avons donc l'intuition que l'utilisation d'une telle phase de pré-production est donc bénéfique vis-à-vis du temps total passé à développer et implémenter la solution. Cela est cependant fort difficile à prouver, au vu de la difficulté à comparer les différentes approches.

En effet, un même travailleur appliquant différentes approches pour un même problème sera vraisemblablement biaisé après l'application de la première approche, celle-ci ayant déjà donné des résultats. Plusieurs travailleurs appliquant différentes approches pour un même problème pose également des soucis – en effet, comment comparer objectivement le degré de compétence des différents travailleurs ?

Enfin, un même travailleur appliquant différentes approches sur plusieurs problèmes différents pose un problème similaire : comment comparer objectivement ces différents problèmes ?

Il ne reste donc, à notre sens, que l'intuition et l'expérience pour se forger un avis sur le débat clé entre la pré-production et la production. Par notre démarche, nous nous sommes convaincus de l'intérêt de la pré-production, et nous allons continuer à l'appliquer pour les problèmes à venir.

5.2.2 Points forts

Le plus gros avantage de cette démarche est de produire un ensemble complet de documents décrivant la modélisation et l'implémentation de la solution. Cela permet donc un processus de correction et de rétro-ingénierie aisé.

De plus, cette démarche permet d'adopter un point de vue plus clair sur la situation, en posant dès le départ l'énoncé complet du problème et l'inventaire des outils à disposition. Ce point de vue donne alors un recul non-négligeable sur le problème, qui permet de réfléchir plus efficacement à des solutions possibles.

Aussi, ce processus permet également de se rendre compte d'emblée si une solution est envisageable ou pas. En effet, le développement par écrit d'une solution aura vite fait de mettre en avant ses imperfections, ce qui permet alors de se rendre compte tôt qu'une solution n'est pas à développer.

Enfin, la modélisation du problème et des solutions, dans son aspect générique, permet de se rendre compte en abordant un autre problème si celui-ci présente des similitudes avec le problème déjà traité, à la manière des *design patterns* de programmation. Découvrir qu'un problème est réductible à un problème déjà traité permet de gagner un temps important sur la conception et l'implémentation des solutions à ce problème.

5.2.3 Points faibles

L'aspect négatif d'une telle démarche est le suivi de la documentation. En effet, si l'énoncé du problème venait à changer, ou si une solution devait être mise à jour, il faudrait également mettre à jour les documents les accompagnant. Cela présente une charge de travail non-négligeable.

De plus, il arrive que le *game designer* derrière l'énoncé du problème se rende compte que ce qu'il a demandé n'est pas cohérent avec sa vision du jeu, et qu'il change alors d'avis en voyant la solution tourner. L'annulation d'une solution à l'étape de prototype implémenté en production coûte alors bien moins cher qu'avec notre démarche, où toute la modélisation aurait déjà été effectuée.

5.2.4 Améliorations

La manière dont nous avons présenté notre démarche convient difficilement pour une consultation et une documentation technique. Il serait intéressant de structurer la démarche plus proche d'un document technique, afin que son suivi soit plus efficace. Ainsi, définir une structure-type pour ce type de démarche et s'y tenir pourrait être fort pratique.

Aussi, pour faire face à d'éventuels changements dans les exigences et pour faire face au problème de « changement d'avis » évoqué dans la section précédente, il peut être intéressant de maintenir un prototype de solution accompagnant la démarche à des étapes-clés, afin de se rendre compte le plus tôt possible d'éventuelles modifications à apporter.

5.3 Conclusion

Notre démarche de recherche est à présent aboutie. Nous avons implémenté et évalué les propositions de solution que nous avions définies, et nous avons mis en évidence les points forts et les points faibles de ces solutions, afin que le *game designer* puisse choisir efficacement la solution qu'il désire conserver.

Ainsi, la solution d'exploration complète présente une grande flexibilité, qui lui sera fort utile si le produit venait à évoluer en proposant un plus grand nombre d'actions et d'indicateurs. La solution de décomposition en sous-recherches, quant à elle, convient mieux à des solutions de taille modeste.

Nous avons également posé un regard critique sur notre démarche de recherche, en mettant en avant ses avantages et ses inconvénients. Ainsi, une telle démarche apporte un recul plus important sur le problème, et produit une documentation plus complète et plus intéressante en termes de réutilisabilité.

Cependant, de tels documents demandent un suivi plus important, et il est moins aisé de se rendre compte tôt qu'une exigence ne convient pas au *game designer*.

Nous avons de même mis en évidence la difficulté de comparer l'approche « pré-production » avec l'approche « production ». Le débat intègre difficilement des composants objectifs. Il repose donc sur l'intuition et l'expérience de convaincre quel aspect privilégier.

Chapitre 6 - Conclusion

Au cours de notre travail, nous avons présenté un problème technique dans un environnement vidéo-ludique, que nous avons tenté de résoudre au travers d'une phase poussée de modélisation et de conception.

Le premier chapitre nous a permis de présenter l'univers de l'industrie du jeu-vidéo, afin de souligner les différences avec l'industrie du logiciel « classique ». Nous avons ainsi décrit les compétences diverses et variées qui interviennent lors de la production d'un jeu vidéo au travers des travaux de (Bethke 2003), en mettant l'accent sur les rôles graphiques, techniques et de conception pour la partie développement. Nous avons également mis l'accent sur le rôle de l'éditeur en tant qu'investisseur dans le projet de développement.

Nous avons ensuite abordé les spécificités du cycle de développement d'un logiciel vidéo-ludique au travers des travaux de (Bates 2004). Nous avons ainsi surtout mis en évidence le rôle de la pré-production, étape-clé entre le concept du jeu, et la mise en production dudit jeu. C'est durant cette étape importante que les éléments qui composeront le jeu sont définis, ainsi que la planification de sa production.

Par après, nous avons abordé les spécificités de l'intelligence artificielle appliquée au jeu vidéo à l'aide des travaux de (Funge 2004). Nous avons ainsi mis en évidence la distinction entre *PC* et *NPC*, tout en explicitant la structure de contrôleur IA. Nous avons également décrit le besoin de traiter chaque problème IA au cas par cas.

Enfin, grâce à l'expérience de (Bethke 2003), nous avons décrit les problèmes majeurs récurrents que l'industrie du jeu vidéo rencontre dans la production de ses logiciels. Nous avons ainsi particulièrement mis en évidence la difficulté à générer du profit à partir d'un jeu, les défauts de planification, et la tension entre pré-production et production.

Nous nous sommes ensuite posé la question de ce que nous pouvions réaliser, en tant que technicien, pour faire face à ces problèmes. Nous avons alors émis l'avis de donner l'importance qu'il se doit à la phase de pré-production, en réalisant un travail de documentation et de modélisation le plus poussé possible pour chaque aspect du développement technique.

Nous avons donc exploré cette idée au cours du second chapitre. Au cours de ce chapitre, nous avons contextualisé un problème d'intelligence artificielle provenant du monde vidéo-ludique en décrivant son environnement, et en définissant formellement son énoncé.

Ce problème concerne une simulation géopolitique du monde actuel. Le but est de modéliser un contrôleur pour les chefs d'état *NPC* afin qu'il décide d'actions à entreprendre afin de résoudre leur déficit budgétaire, en se basant sur plusieurs variables d'environnement telles que le montant du déficit, la popularité du chef d'état, et son alignement politique envers son parti.

Nous avons ensuite entrepris la démarche de résoudre ce problème à l'aide d'une modélisation et d'une conception poussée. Nous avons ainsi établi une solution type au problème, en le ramenant à un problème d'optimisation.

Par après, nous avons exploré deux propositions de solution instanciant la solution type. La première consiste à considérer l'entièreté de l'espace des solutions. Cet espace étant potentiellement très grand, le parcourir en entier serait fort coûteux. Nous avons donc choisi d'appliquer une heuristique, l'algorithme du Tabou, afin de réduire la complexité de la recherche, au coût d'une précision moins importante.

La deuxième solution proposée consiste à diriger la recherche en évaluant chaque action et en ne considérant que la meilleure à chaque étape. Cette solution est moins complexe et plus facile à mettre en œuvre, mais ignore un grand nombre de solutions possibles. Elle compte donc ainsi sur la direction donnée par les meilleures actions unitaires afin de trouver une solution satisfaisante.

Le troisième chapitre a abordé l'implémentation de ces solutions. Nous avons d'abord d'écrit l'environnement du logiciel cible de l'implémentation, le jeu *Rulers of Nations* développé par *Eversim*. Nous avons donc ainsi présenté le studio de développement et les grandes lignes directrices du jeu. Nous avons ensuite présenté l'architecture logique du jeu au travers de ses composants majeurs, avant de décrire les principales structures de données dont le jeu fait usage.

Ensuite, nous avons décrit nos choix d'implémentation quant à la valeur des constantes et à la structure des variables de la modélisation. Nous avons alors entrepris de parcourir chaque spécification que nous avions définie afin de présenter nos choix d'implémentation les concernant.

Le dernier chapitre fit l'état de l'évaluation des solutions proposées et de la démarche qui a mené à la définition de ces solutions. Nous avons donc ainsi évalué les deux solutions sur base des critères définis dans le second chapitre, tout en mettant en avant leurs points forts et leurs points faibles. Nous avons ensuite suggéré des améliorations pour ces solutions, avant de les comparer sur leurs caractéristiques majeures.

De cette évaluation est ressorti que les deux solutions convenaient, mais se différenciaient sur deux points importants : la flexibilité et la convergence. En effet, la solution de l'exploration complète propose une flexibilité important au niveau de la précision et du temps de calcul, au vu des paramètres de l'heuristique utilisée. La seconde solution est beaucoup plus rigide, celle-ci devant toujours parcourir l'ensemble des actions.

Au niveau de la convergence, la deuxième solution est cependant la mieux placée. En effet, à toujours considérer la meilleure action possible en premier lieu, elle donne des résultats plus rapidement. Cependant, elle risque de très vite rencontrer un problème de charge de calcul avec l'augmentation de la taille du problème, principalement vis-à-vis du nombre d'actions.

Quant à notre démarche, nous nous en estimons satisfait. Elle débouche ainsi sur une documentation importante et fort pratique pour de la rétro-ingénierie et la réutilisabilité. Elle pose cependant une charge de travail supplémentaire pour assurer le suivi vis-à-vis des corrections et mises à jour de l'énoncé et des solutions.

L'étape suivante est de structurer la recherche plus comme un document technique standardisé, afin d'adopter un niveau de clarté de consultation plus élevé, ainsi qu'une résilience plus importante aux changements potentiels d'exigences ou de conception.

Nous espérons avoir montré au travers de notre démarche les qualités d'une pré-production poussée, par opposition à une phase de production prenant le pied trop tôt sur celle-ci.

Bibliographie

- Bates, Bob. *Game Design (2nd ed.).* Boston: Thomson Course Technology, 2004.
- Bethke, Erik. *Game development and production.* Plano: Wordware Publishing, Inc., 2003.
- Funge, John. *Artificial intelligence for computer games: an introduction.* Wellesay: Peters, 2004.

Annexe A - Interviews

Cette annexe reprend les différentes interviews que nous avons conduites au cours de notre expérience au sein du studio *Eversim*. Ces interviews concernent le développement de l'intelligence artificielle au sein du produit *Rulers of Nations*.

Toute description du logiciel qui se trouve dans les interviews est confidentielle, les droits sur ce contenu appartenant à la société Eversim.

A.1 Clément Laugraud

Cette interview a été conduite auprès de Clément Laugraud, programmeur stagiaire responsable de l'IA du mode compétition de *Rulers of Nations*. En voici le résumé.

Quelles ont été les tâches qui t'ont été confiées, et combien de temps as-tu eu et as-tu pris pour les réaliser ?

Clément : Cela fait 6 mois que je travaille sur l'IA du mode compétition. Au départ, le plan était d'avoir plusieurs IA séparées, indépendantes : une pour les contrats, une pour la politique nationale, une pour l'internationale, etc.

J'ai tout d'abord eu la tâche de développer l'IA des contrats – c'est-à-dire, l'IA qui se charge de passer des contrats de vente et d'achat avec d'autres pays en cours de partie. Pour ce faire, l'équipe des Game Designers m'a transmis une série de documents décrivant comment devait se comporter l'IA – sortes d'arbres de décisions qui décrivaient le comportement de l'IA selon tel ou tel événement.

Ces documents comportaient malheureusement des erreurs, et étaient très basiques. J'ai donc pris la liberté de les améliorer en mettant plus d'intelligence dans la fabrique de contrats, comme prendre en compte l'offre et la demande, l'alignement des pays envers la nation de l'IA, etc.

Ces modifications, je les ai présentées après coup aux Game Designers, qui fort heureusement les ont validées. Cette tâche fut terminée au bout de 2 mois, incluant le temps qu'il m'a fallu pour m'habituer au logiciel et à ses sources.

Ma deuxième tâche fut de développer l'IA gérant la politique nationale. A cette fin, il m'a également été transmis différents documents de Game Design. Cependant, lors du développement, la décision fut prise d'intégrer toutes les IA en une et une seule entité. J'ai donc dû changer mes plans afin d'intégrer toutes ces IA dans celle que j'étais en train de développer.

Ainsi j'ai dû également désactiver l'IA qui gérait les contrats et l'intégrer dans l'IA générale, ce qui fut aisé grâce au fait que toutes mes fonctions étaient déjà présentes et fonctionnelles. Cette tâche me pris un peu plus de 3 mois de travail.

La dernière tâche fut d'intégrer un mécanisme de "vol de contrats", où une IA annulerait ses autres contrats concernant la même denrée si un autre pays venait à lui offrir la même chose à un prix moins important. Cette tâche est toujours en chantier, mais les mécanismes de base sont déjà fonctionnels.

Quelles aides t'es-tu rédigées pour t'accompagner dans ton développement ?

Clément : Tout d'abord, j'ai découpé le travail en tâches et sous-tâches, et réalisé un planning estimant le temps de travail nécessaire à la réalisation de celles-ci. Ces tâches comportaient ce qui était demandé ainsi que mes suggestions, et le tout classé par ordre de priorité. Ce planning fut également d'intérêt pour le management, étant données les échéances.

J'ai ensuite rédigé un document reprenant les codes des actions que j'aurais à utiliser et leur signification, afin de m'aider à comprendre les sources et à intégrer mes modifications.

Afin d'implémenter efficacement ces dernières, je me suis aidé de schémas UML et d'ordinogrammes tracés à la main. Enfin, pour tester l'équilibre des différentes IA, j'ai composé un document chargeant les statistiques de parties lancées en boucles et permettant de comparer les résultats.

As-tu eu l'occasion de participer au design des IA ?

Clément : Oui et non. A savoir que les documents m'étaient donnés tels quels, mais que mes suggestions étaient toujours entendues ; j'ai donc pu intervenir dans le processus et garnir les IA d'une intelligence moins basée sur le simple aléatoire.

Que penses-tu des échéances qui t'ont été données ? As-tu pu les respecter ?

Clément : Les échéances me semblaient correctes vis-à-vis de la charge de travail. Cependant, elles n'incluaient en rien le temps nécessaire au test et au debugging, ce qui a malheureusement fait qu'elles ont été dépassées.

Le problème vient du fait qu'on s'intéresse surtout au résultat : il faut que ce soit réalisée et ce le plus vite possible ; peu importe les moyens. Au bout du compte, les tâches ont quand même été réalisées et les IA tournent de manière satisfaisante, bien qu'il reste encore beaucoup d'aspects à améliorer, à mon goût.

Qu'est-ce qui te paraît bon et moins bon dans les IA qu'on t'a demandé de développer ?

Clément : Leur processus décisionnel de base est trop simple : un arbre simple, déterministe, avec peu de conditions. Il serait plus approprié pour un jeu de simulation réaliste d'avoir une intelligence plus riche et plus profonde, intégrant plus de paramètres.

En effet, il est important que les comportements des nations non-joueuses soient crédibles afin de maintenir l'immersion, ce qui n'est pas toujours mon sentiment dans l'état actuel des choses.

Une amélioration possible serait, par exemple, d'utiliser des algorithmes génétiques, ce qui donnerait une plus grande largeur et profondeur aux IA. Le problème serait alors la paramétrisation plus lourde et la perte du déterminisme, qui peut encombrer le processus de test et de debugging.

Les ressources ne manqueraient-elles pas en ce cas ?

Clément : Pas avec une bonne paramétrisation ! Il est possible de donner des réactions plus ou moins abouties et recherchées en fonction des ressources et du temps disponibles.

As-tu été satisfait de ton contact avec l'équipe de Game Design ?

Clément : Les informations qui m'étaient données étaient parfois précises, parfois vagues ; il me fallait donc régulièrement que je complète ou corrige par moi-même certains aspects. Cela m'a par contre donné l'opportunité de participer au design des IAs et me faisait jouir d'une grande flexibilité dans mon travail, ce qui était un confort non-négligeable.

As-tu été satisfait du résultat de ton travail ?

Clément : Dans l'ensemble oui, malgré le fait que je sente qu'on peut en faire beaucoup plus. Je déplore seulement le manque de temps qui m'a été alloué pour tester et débugger, ainsi que le manque de documentation du code source original, ce qui m'a coûté beaucoup de temps à évaluer comment intégrer mon propre travail à l'existant.

Je crois qu'on a fait le tour. Merci beaucoup pour ton temps et ta collaboration !

A.2 Xavier Marot

Cette interview a été conduite auprès de Xavier Marot, chef de projet et *Lead Game Designer* au sein de l'équipe de développement de *Rulers of Nations*. En voici le résumé.

Comment s'est construit le système d'IA du jeu Rulers of Nation ?

Xavier : le système a été développé par étapes. En premier lieu, nous avons implémenté une IA dite nationale, c'est-à-dire des réactions aux (in)actions du joueur. Ces réactions se déclenchent lorsque certaines conditions sont remplies, soit par un appel direct à un « pool » de réactions où le programme filtrera celles qui doivent effectivement sortir sur base de ces conditions en question, soit par le biais de « permanents » qui évaluent ponctuellement des paramètres particuliers afin de décider de la sortie d'une réaction.

Un des points importants de ces paramètres sont les thèmes. Chaque groupe et personnalité importante est caractérisé par des thèmes qui représentent son orientation, et définissent sa réaction face aux actions du joueur.

Par exemple, taxer les propriétaires d'armes va mécontenter l'association des chasseurs, mais va contenter le groupe écolo. Cela aide aussi à définir la position du parlement sur les lois à voter, le déclenchement de grèves lorsque les syndicats sont mécontents, etc.

Tout ceci sert d'indicateurs, de marqueurs pour aider le joueur à orienter sa politique, et fut donc la base de notre système IA.

Ensuite est venue se greffer l'IA internationale - en d'autres termes, l'IA des chefs d'états non-joueurs. Le premier point était de faire en sorte que ceux-ci donnent une réponse aux requêtes « à questions ». Le comportement par défaut a été de toujours répondre négativement, puis nous l'avons affiné en donnant des critères à certaines réactions importantes.

Le deuxième point fut l'IA du wargame, principalement les décisions autour de la gestion des unités (quoi, où et quand envoyer des unités), ainsi qu'autour des traités.

Le dernier point fut l'IA des contrats et du jeu de compétition. L'IA des contrats décide de la proposition et de la négociation de contrats, tandis que l'IA du jeu de compétition décide des actions qu'un pays non-joueur parmi les 16 participants entreprendra.

Cette IA du mode compétition fut particulièrement différente de nos autres implémentations, tout d'abord parce-que cette IA se concentre sur les objectifs de victoire de la partie plutôt que le réalisme, ensuite parce-qu'elle repose sur un pool d'actions qui change en fonction du comportement attribué à l'IA au lieu d'un script bien défini.

De quelle manière l'implémentation de l'IA du mode compétition a-t-elle été gérée ?

Xavier : en tout premier lieu, nous avons mené un travail de design de l'architecture de la base des actions, et de définition des comportements que peuvent adopter les IA. La seconde étape fut de programmer la lecture de la base dans le logiciel, puis de faire usage des données ainsi récupérées. La dernière étape, itérative, fut organisée autour de tests et d'ajustements axés sur deux points.

Le premier fut de tester la qualité de la structure, à savoir l'absence de bugs, l'adéquation aux spécifications, l'apport au gameplay, etc.

Le deuxième point fut l'équilibrage des actions et des comportements, voire des nations. En effet, il ne fallait pas qu'un comportement particulier l'emporte sur les autres, ni que la nation la plus performante soit toujours la même.

Pour tester cet équilibre, nous avons fait usage de tests automatiques, à savoir des parties tournant sur des boucles de 4 ans du jeu et produisant des statistiques que nous pouvions synthétiser afin d'avoir une vue sur la performance des comportements et des nations.

Quelles contraintes majeures avez-vous rencontrées lors du développement du système IA ?

Xavier : tout d'abord des contraintes de temps. Dans son premier design, l'IA du mode compétition par exemple ne comportait qu'une petite quantité d'actions scriptées, faute de pouvoir élaborer.

La sortie du jeu ayant été repoussée, cela nous a permis de rendre le système plus complexe, pour devenir ce qu'il est aujourd'hui. De fait, nous sommes allés au-delà d'un système juste nécessaire, en développant des enrichissements.

La deuxième contrainte importante fut la performance. Cela était plus une inquiétude qu'une contrainte : il fallait que le système tourne tel que prévu, mais sans ralentir le jeu de manière notable.

Cependant, le nombre peu important de calculs dû au fait que seuls 16 pays potentiels sont gérés par l'IA, ainsi que la lecture complète de la base des actions en mémoire en début de partie ont fait que la performance ne nous a pas posé d'inconvénient.

Pensez-vous qu'il serait intéressant d'ajouter ou d'élaborer des fonctionnalités du système ?

Xavier : il est évident qu'on peut faire plus, par exemple en étendant le modèle de l'IA du mode compétition à toutes les nations en mode simulation. En effet, un des problèmes du jeu à ce niveau est que le joueur est seul déclencheur d'événements : mis à part quelques actions scriptées, le monde ne fait que réagir aux actions du joueur.

Ensuite, nous pourrions améliorer le système de comportements : rajouter plus d'actions et de comportements, ajouter une pondération plus fine des actions afin que l'IA détermine la meilleure chose à faire à un instant donné, voire complexifier les buts en apportant des objectifs composites de plus haut niveau, de sorte à ce que le joueur ne puisse pas percevoir clairement les intentions de l'IA.

Il semble que nous ayons fait le tour. Merci beaucoup pour votre collaboration !

Annexe B - Journal d'expérience

Au cours de notre expérience au sein du studio de développement *Eversim*, nous avons tenu un journal qui faisait état de notre prise en main du produit et de la réalisation de nos tâches sur celui-ci.

Ce journal capture les tâches que nous avons dû réaliser durant notre stage de fin d'études, qui a pris place d'août 2010 à janvier 2011 auprès de la compagnie Eversim basée à Lognes, en France.

Il correspond au relevé régulier des travaux, de nos impressions et de notre découverte du logiciel étudié, à savoir le jeu Geo-Political Simulator 2 : Rulers of Nations.

Il est structuré en entrées journalières, qui recensent chaque travail et résument leur résolution, hebdomadaires, qui résument chaque semaine de travail en donnant les impressions et explications jugées pertinentes, et mensuelles, qui résument brièvement le travail effectué au cours du mois.

Toute description du logiciel qui se trouve dans les entrées est confidentielle, les droits sur ce contenu appartenant à la société Eversim.

B.1 Rapports mensuels

Ces rapports résument brièvement le travail et la découverte accomplis chaque mois du stage.

B.1.1 Août

Ce premier mois de stage me permit de faire mes premiers pas au sein de l'entreprise. L'équipe étant fort amicale, j'ai pu m'intégrer aisément dans celle-ci. Le code étant fort peu documenté et créé sans design au préalable, il fut difficile de l'appréhender, mais j'y suis maintenant relativement bien habitué. Quant aux outils (Visual Basic, SVN), je les maîtrise à présent correctement.

Ma démarche de debugging est maintenant assez bien rodée :
- Identifier l'entité responsable à l'aide de la console ;
- Identifier la méthode ou l'événement potentiellement responsable à l'intérieur de l'entité ;
- Placer des points d'arrêts dans ces emplacements ;
- Exécuter le code pas à pas pour détecter l'endroit où l'erreur se produit ;
- Corriger l'erreur.

J'ai pu également apprendre le fonctionnement et la structure du logiciel :
- S'appuie sur une librairie propre gérant les opérations de bas niveau, la GEOL ;
- S'articule autour d'un système d'événements, agissant comme des pushes sur les entités ;
- Se construit à la manière d'un arbre, chaque entité ayant un parent et des enfants ;
- Fait forte utilisation de variables globales ;
- Limite grandement l'utilisation de classes ;
- S'appuie sur des getters spécifiques (les getvals) pour obtenir des informations qui sont structurées dans des variables globales ;
- Rassemble et lit ses informations inertes dans des bases de données enregistrées sous forme de fichiers texte.

B.1.2 Septembre

Ce deuxième mois fut également un mois axé sur le debugging de l'application. Mon processus d'identification et de correction de bug est plus robuste, et j'ai dû m'adapter à de nouvelles situations, et parfois impacter plus profondément le code. C'est également fin de ce mois qu'est sortie l'application, ce qui nous fais nous concentrer sur le patching plus que sur le développement à présent.

Du côté du neuf, j'ai pu être initié à :
- la structure de l'IA dans le mode compétition ;
- l'utilisation des bases de données (action et réaction surtout) ;
- l'utilisation de triggers et la gestion des actions.

J'ai également pu faire le point et être un peu plus conscient des défauts de structuration qui ont menés aux problèmes que rencontre l'application aujourd'hui, à savoir une absence de structuration du processus de développement, causé par la pression du manque de ressources (temps, argent, effectifs), et sans doute par un manque de prise de conscience de la nécessité et de l'importance d'une telle structuration.

Enfin, j'ai pu faire le point sur l'ensemble des IA présentes dans le jeu, à savoir :
- les chefs d'états non-joueurs, réactifs en mode simulation et pro-actifs en mode compétition, et leur système de négociation de contrats axé sur la maximisation de leur utilité tout en conservant celle du joueur ;
- le peuple, les personnalités et les syndicats avec leurs réactions basées sur des critères face aux actions du chef d'état, et leur fidélité à ce dernier qui peut déclencher diverses réactions ;
- les unités en mode Wargame avec leur Pathfinding.

B.1.3 Octobre

Ce mois fut quelque peu difficile au départ, du fait que je sois toujours affecté à des tâches de debugging et l'envie étant plutôt au développement. Cependant, la perspective d'améliorer l'expérience de jeu des joueurs en corrigeant les problèmes existants m'ont donné une toute nouvelle motivation. De plus, j'ai eu l'occasion d'apporter certaines améliorations au système existant, tel que l'ajout d'une coloration spéciale dans le journal pour les articles entraînant une variation de popularité.

Aussi, j'ai pu travailler plus en profondeur avec le moteur économique. Celui-ci est complexe et large, et malheureusement le programmeur qui s'est chargé de le constituer ne fait plus partie de l'équipe, ce qui pose des problèmes importants quand il s'agit d'y apporter des corrections ou des améliorations. Mon travail ici fut de réintégrer et de corriger la rubrique de la monnaie du pays, avec en particulier la mécanique du taux directeur.

Les travaux majeurs de ce mois furent donc :

- La correction du calcul du taux d'espérance de vie ;
- La coloration d'articles dans le journal ;
- La refonte du calcul des probabilités de réussite d'un assassinat ;
- La réintégration de la fenêtre Monnaie, dont la possibilité de rejoindre/quitter l'Euro et de changer le taux d'intérêt directeur du pays.

B.1.4 Novembre

Ce mois fut particulièrement prolifique au niveau des travaux sur le moteur économique. En effet, j'ai eu fort à faire au niveau de la gestion des devises, l'évolution de leur valeur au cours du temps ayant été récemment réintroduite.

La notion importante à retenir pour celles-ci est la conversion. De fait, tous les montants du jeu sont enregistrés et manipulés en interne dans une devise unique dont la valeur ne change pas : le dollar moteur. Il y a donc conversion constante entre montants destinés à l'affichage dans la devise de l'état et les montants destinés à être manipulés en interne.

En plus de cela, on trouve une conversion entre les différentes devises du monde, qui sert à traduire une masse monétaire d'un pays vers la masse monétaire de l'autre.

Après les devises, ce fut le tour de la balance commerciale, dont le calcul était incohérent. A noter qu'elle fait partie des valeurs dites lissées, c'est-à-dire qu'elle correspond à un mélange entre la valeur de l'exercice précédent et la valeur de l'exercice en cours, cette dernière étant un cumul effectué à chaque tour économique et donc incomplète jusqu'à la fin de l'année en cours.

Enfin, le dernier morceau compliqué fut les allocations familiales. Celles-ci, comme les autres lois, sont enregistrées dans une structure d'évolution (un evol). Cependant, cette structure ne possède la place que pour une seule valeur, alors que les allocations concernent 3 catégories d'enfants. L'implémentation initiale était donc de couper la place mémoire en 3 pour retenir ces 3 valeurs dans un seul espace. Cela a montré ses limites, principalement dûes aux degrés d'arrondis importants et posant de gros problèmes dans certaines devises. J'ai donc refondu le système pour travailler avec 3 evols plutôt qu'un seul.

B.1.5 Décembre

La première partie du mois de décembre me vit travailler sur la limitation de la variation de la valeur des devises du monde. En effet, celles-ci atteignaient parfois des taux de change astronomiques, et cela a été corrigé en introduisant des bornes et une gestion assez basique du taux directeur par l'IA.

Ces lacunes dans l'équilibrage du budget de la part de l'IA me firent imaginer une IA économique qui chercherait à rendre et à garder le budget d'un état viable, afin d'éviter des incohérences à long terme dans le moteur. Cette tâche est cependant ardue du fait du nombre importants de variables et de la difficulté de prévoir des effets économiques avec précision.

Il m'a également été permis de me familiariser un peu plus avec la structure globale du moteur économique. A noter que celui-ci traversa deux étapes ; la première donna lieu au moteur non-contraint, qui laissait l'évolution naturelle dicter les résultats (optique bottom-up). Cette étape échouera du fait de la difficulté à contrôler et à stabiliser le moteur. Ces inconvénients seront palliés en contraignant le moteur, ce qui fut la deuxième étape. Dans le moteur contraint, ce sont des prévisions qui dictent les résultats (optique top-down).

Enfin, j'ai également pu commencer une tâche de développement un peu plus conséquente, qui consiste en l'introduction d'un nouveau système de construction, apportant le parallélisme au lieu de la simple succession de chantiers.

B.1.6 Janvier

La première partie de ce dernier mois de stage fut consacrée à la fin du développement du nouveau système de construction. Ce développement rencontra quelques difficultés, notamment au niveau des structures d'évolution qu'il a fallu modifier afin de pouvoir sauvegarder divers paramètres nécessaires à la mise à jour des données de la nation une fois la construction terminée.

La partie la plus importante fut dédiée au développement d'une nouvelle fonctionnalité, censée sortir avec la prochaine version du jeu, d'ici moins d'un an. Cette fonctionnalité consiste à offrir au joueur la possibilité de nationaliser et de privatiser des secteurs d'activité, afin d'obtenir plus de contrôle sur une activité ou, au contraire, de lâcher du lest afin de réaliser une rentrée rapide d'argent.

Le moteur économique étant souvent perçu comme black-box dans la culture de l'entreprise, je me suis donc efforcé de rendre l'implémentation de la fonctionnalité la plus black-box possible, en incluant la plupart des méthodes dans la structure d'un secteur d'activité et en fournissant une documentation riche.

Le développement fut ponctué de réunions avec le management afin de fixer plus précisément les spécifications et de faire face à des problèmes nouveaux qui ont pu être mis en évidence par le système.

La fonctionnalité livrée, j'arrivai au bout de mon stage. Ce dernier, bien que ne portant pas directement sur l'objet annoncé, fut cependant toujours pertinent et très enrichissant, tant au point de vue compétences que social, le personnel m'ayant laissé une fort bonne impression.

B.2 Rapports hebdomadaires

Ces rapports constituent un résumé des travaux, observations et impressions jugés pertinents dans le cadre du travail.

*B.2.1 Semaine 1 (*02/08/2010-06/08/2010*)*

Cette semaine marqua mes premiers pas dans l'entreprise, et son but fut surtout de faire plus ample connaissance avec l'équipe et de s'habituer au code ainsi qu'aux mécanismes internes de la société. L'équipe de développement étant très amicale, mon intégration fut relativement aisée ; qui plus est, la moitié de l'équipe est composée de stagiaires, ce qui facilita d'autant plus la tâche. Le code est, quant à lui, assez difficile à appréhender. En effet, sa documentation est assez pauvre, parfois inexistante, et il est le fruit des efforts combinés de programmeurs qui se sont succédés sur ses différents aspects, apportant chacun leur manière de coder. Le résultat en est donc par moment problématique, et j'ai l'impression que l'entreprise paie ce défaut à cette époque proche de la fin du développement, où les bugs sont nombreux et les délais sont proches d'être dépassés.

Les mécanismes internes de l'entreprise supportent assez bien le travail de développement, avec comme outil principal un "Task Manager" qui permet aux employés de créer, modifier, attribuer, vérifier et valider des tâches et ce très aisément grâce à l'intranet de l'entreprise. Les sources sont, quant à elles, synchronisées grâce à un serveur SVN interne. Un outil de pointage électronique permet d'enregistrer ses heures de travail et de visualiser notre décompte du mois, ce qui nous donne une certaine flexibilité dans nos horaires – tout ce qui importe étant que nos heures soient faites.

*B.2.2 Semaine 2 (*09/08/2010-13/08/2010)

Après deux semaines, je navigue plus facilement dans le code. Après avoir passé quelques dizaines d'heures à étudier plusieurs de ses aspects, je commence à trouver certains automatismes qui peuvent faire gagner beaucoup de temps.

Le code s'appuie sur une librairie, la GEOL, reprenant une série de fonctions de bases, comme des fonctions graphiques ou de traitement de chaînes de caractères. La structure principale du logiciel est un système à événements : dans chaque entité tourne une boucle qui est appelée à chaque "frame" du jeu, et qui vérifie la présence d'événements qu'une autre entité lui aurait fait parvenir. Cet événement est typé et comporte éventuellement des paramètres, ainsi que l'adresse de son émetteur. Certains événements proviennent de la GEOL, et la plupart sont créés par les programmeurs, selon leurs besoins.

Les entités du code sont articulées selon un système parent/enfant, chaque entité ayant un parent (à l'exception de l'entité principale), et éventuellement un certain nombre d'enfants. Comme dans un arbre, si une entité est détruite, tous ses enfants sont détruits avec elle.

Une autre caractéristique importante du code est sa forte utilisation de variables globales. Celles-ci sont définies par centaines dans un fichier source, accompagnées de toutes les constantes et structures définies par les programmeurs. Cela cause souvent des conflits, certaines méthodes lisant et modifiant des variables en même temps.

Enfin, l'utilisation de classes est très limitée. Toute entité est déclarée par un mot-clé prédéfini, et ne comporte que des attributs privés et des méthodes parfois privées, souvent publiques et statiques. Leur fonctionnement interne s'appuie très fortement sur le système des événements décrit plus haut.

*B.2.3 Semaine 3 (*16/08/2010-20/08/2010)

La 3e semaine fut sans doute la plus productive jusqu'à présent. Je commence à maîtriser assez bien l'IDE que l'entreprise utilise, à savoir Visual Basic, et j'ai plus souvent la bonne intuition pour localiser un problème dans le code.

Au début, j'utilisais surtout les points d'arrêt, qui permettaient d'arrêter le logiciel une fois une certaine ligne de code atteinte. Je suivais ensuite pas à pas l'exécution des instructions, jusqu'à, dans le meilleur des cas, tomber sur l'erreur. Cette méthode, utilisée telle quelle, s'avère lente. Elle fut cependant bien utile pour comprendre le fonctionnement du logiciel.

Ensuite, je fis usage de la console fournie au sein du logiciel. Cette dernière permet d'afficher les entités en cours de vie, ce qui rend l'identification de celles intervenant dans une situation à problème beaucoup plus aisée et bien moins aléatoire. En effet, avant cela il m'arrivait même d'essayer de deviner le nom d'un objet ou d'une fonction qui pourrait avoir à faire avec l'erreur.

Après m'être habitué aux événements et en particuliers aux récurrents tels que ceux envoyés lors du clic sur un bouton, maintenant je procède de manière plus efficace : j'identifie l'objet potentiellement responsable de l'erreur à l'aide de la console, je cherche sa déclaration dans le code, j'identifie l'événement qui est intéressant à étudier, et j'y place un point d'arrêt de manière à l'analyser plus en détails.

Cette méthode porte relativement bien ses fruits, tirant profit de la structure restée très simple du logiciel.

*B.2.4 Semaine 4 (*23/08/2010-27/08/2010)

Cette semaine, j'ai pu m'intéresser plus en détail au fonctionnement du système réseau mis en place à l'intérieur du jeu. En effet, afin de permettre le jeu multi-joueurs, une refonte assez importante du logiciel a dû être opérée lorsque l'ajout de cette fonctionnalité fut décidé.

Comme dit précédemment, les entités vont principalement chercher leurs informations dans des variables globales, initialisées en début de partie, et mises à jour au cours de celles-ci. On retrouve par exemple dans ces dernières les différents personnages, nations, groupes et organisations, les rendez-vous, les sommets à venir, etc.

Dans un jeu en réseau, on retrouve le serveur qui détient toutes ces informations, et les clients qui ne sont qu'une simple vue sur le jeu. Ceux-ci n'ont donc plus accès à ces variables globales, et doivent donc constamment en faire la demande au serveur.

C'est ainsi qu'est né le système des "Getvals". Ces méthodes s'occupent d'aller chercher des informations chez le serveur sous forme de structures prédéfinies (les vals). On retrouve des vals pour des aspects particuliers du jeu (val_nation, val_perso, …), ainsi que des vals spécifiques à des entités (val_ask_action, …). Quand un client a besoin d'informations, il doit donc en faire la demande au serveur par un "getval", et récupèrera la valeur par un push du serveur ou un pull par lui-même selon le mode. Les vals sont, bien sûr, également des variables globales.

Ce système a été mis en place même pour les parties hors ligne (jeu en solo), ce qui a donc modifié des appels à beaucoup d'endroit du code. Cela a été la cause de nombreux bugs, encore apparents aujourd'hui. Avec ce système a également été mis en place un mécanisme similaire, mais cette fois-ci gérant les actions. Une fois qu'un client donne un ordre, il envoie cet ordre au serveur pour que celui-ci le répercute. Ce mécanisme est également en place même dans les parties hors ligne.

Ces modifications qui sont intervenues au milieu du développement ont laissé en de nombreux endroits des méthodes qui n'ont pas été mises à jour, et cela est la cause de nombreux problèmes. Il m'arrive bien souvent de devoir corriger l'appel à un getval, son utilisation, ou même le getval lui-même.

Je profite du fait que je parle de la gestion des informations pour aborder les bases XLS. Les bases XLS sont des bases de données contenant des textes, des identifiants et des conditions propres à de nombreux mécanismes du jeu. On retrouve les intitulés (et leur traduction) des interfaces, les textes des personnages, requêtes et réactions, etc. Ceux-ci sont lus directement dans les bases grâce à un "gettext", qui, sur base d'un label, d'un offset, de la langue et du nom de la base va retrouver le bon texte dans celle-ci.

D'autres bases sont utilisées pour définir les conséquences d'une action, que ce soit un événement créé par le joueur en lui-même ou par le jeu (comme une catastrophe naturelle). On retrouve également les informations sur toutes les nations, comme leur démographie, leurs villes, leurs personnages, etc.

*B.2.5 Semaine 5 (*30/08/2010-03/09/2010)

Le programmeur responsable de l'IA arrivant en fin de stage en ce début de semaine, il a été convenu qu'il me mette au fait de comment l'IA actuelle était programmée.

Il existe 5 types de comportement pour une IA en mode compétition : conquérante, populaire, économiste, fourbe et diplomate. Une IA conquérante cherchera à développer ses forces militaires et à attaquer ses voisins les plus faibles et qu'elle aime le moins. Une IA populaire fera tout son possible pour augmenter son score de popularité. Une IA économiste aura comme but de réduire le plus possible son déficit budgétaire à l'aide de réformes économiques. Une IA fourbe fera en sorte de gêner le plus possibles les joueurs les mieux classés, en volant leurs contrats et en les infiltrant. Enfin, une IA diplomate fera son possible pour améliorer ses relations avec les pays étrangers.

En début de partie, 3 valeurs sont tirées pour chacune des IA : une entre 0 et 20, une autre entre 20 et 40, et la troisième entre 40 et 60. Ces nombres seront affectés à des comportements et les pondèreront.

A chaque "tour" de l'IA (un nombre de jours de jeu variable selon la difficulté), elle va réévaluer sa stratégie, i.e. son comportement, selon des critères définis dans la base de données IA. Les pondérations tirées en début de partie interviennent ici pour pousser des IA à suivre leur comportement "favori" (celui avec le plus haut score). Par exemple, une IA conquérante risque de perdre de la popularité en attaquant des pays voisins ; elle peut donc passer par une phase de pure campagne populaire avant de reprendre les armes. A remarquer qu'il existe autant un critère pour adopter une stratégie qu'un critère pour en sortir. Aussi, si une IA ne remplit aucune condition d'adoption de stratégie, elle choisira la populaire par défaut.

A propos de la base IA, celle-ci est structurées en Stratégies (les comportements), qui comportent des Groupes d'actions, ceux-ci comportant des actions élémentaires. A chaque tour, une IA va effectuer une action dans chaque groupe de sa stratégie. A chaque groupe est assigné un budget, en % du PIB de la nation de l'IA. Ce budget représente la somme que l'IA va investir dans les actions de ce groupe. Les actions, elles, possèdent des critères et une fréquence en jours. Si les conditions d'une action ne sont pas respectées, l'IA cherchera à réaliser l'action suivante dans la liste. Si elle épuise les actions du groupe, elle ne fera rien pour ce groupe ce tour-ci.

Il est important d'ajouter que le programmeur IA (Clément Laugraud) a pris grand soin d'implémenter l'IA de la manière la plus propre possible, selon ses moyens en temps. Bien que le design présente des défauts du fait d'exigences parfois peu claires, du manque de temps et de l'existant déjà défectueux, il reste aisément compréhensible, ce qui devrait grandement me faciliter la tâche.

Concernant mon travail, je suis toujours occupé à débugger le logiciel existant au jour le jour. J'avoue que ces 5 semaines passées à réaliser la maintenance d'un logiciel sur lequel je n'ai pas travaillé commencent à affecter ma motivation ; cependant, je devrai d'ici peu reprendre le travail de Clément sur l'IA, ce qui devrait alors rendre mes tâches plus intéressantes.

*B.2.6 Semaine 6 (*06/09/2010-09/09/2010*)*

Il est à remarquer que pour les week-ends où je rentre en Belgique, je quitte Paris le jeudi soir, ce qui implique que certaines semaines ne comportent que 4 jours de travail. Cette semaine-ci fut la première dans le cas. Cet avantage pratique vient du fait que vu que je preste des heures "en plus" les autres semaines (i.e. plus de 35 heures), il m'est possible de les récupérer en congés. Je prends ceux-ci les week-ends de retour, pour raisons pratiques.

Cette semaine fut assez monotone au niveau de l'apprentissage. Je suis toujours affecté à la correction des bugs de l'application, les rapports de ceux-ci arrivant régulièrement. Je serai donc encore affecté au debugging pour plusieurs semaines.

Un des bons côtés de cette semaine fut mon amélioration de l'interface réseau, qui m'a sorti de la routine de corriger ce qui ne marche pas pour me mettre à analyser et à améliorer un système fonctionnel. Se sentir "propriétaire" d'une partie du logiciel donne un tout autre sentiment d'accomplissement, bienvenu dans ce cas-ci. J'essaie donc au possible d'entrecouper des tâches de correction avec des tâches d'amélioration, ce qui peut parfois être compliqué au vu des délais actuels. En effet, étant en plein dans le mois de sortie du jeu, le temps presse pour corriger tout ce qui peut l'être avant le gong final, et la priorité va donc à la correction plutôt qu'à l'amélioration.

*B.2.7 Semaine 7 (*13/09/2010-16/09/2010*)*

L'échéance de sortie du jeu approchant à grand pas, le temps est aux heures supplémentaires pour s'assurer que le logiciel soit le plus stable possible. Il est cependant malheureux que beaucoup de problèmes subsisteront à la sortie, ce qui fait qu'un patch à la sortie et un dans le mois suivant sont déjà prévus.

A ce propos, il m'est d'avis que la grande majorité des problèmes que l'on rencontre avec le logiciel en production vient du fait que les phases de modélisation haut et bas niveau ait été laissées de côté. Les raisons évoquées à ce sujet indiquent des délais courts et des moyens réduits, qui empêcheraient de pouvoir "s'attarder" à définir une structure formelle avec les documents nécessaires à une production propre, et/ou à intégrer dans l'équipe un responsable se chargeant de la validité et de l'homogénéité du design. Il me semble cependant que le temps et l'argent "perdus" à mettre en place ce genre de mesures seraient largement compensés par des phases de test plus efficaces et une maintenance beaucoup moins conséquente et plus rapide.

Un des plus gros problèmes visiblement, est qu'il faut "bien commencer" un projet, ce qui n'est pas toujours évident voire possible quand on doit supporter les premiers mois de développement sans aucune rentrée d'argent, ce qui fait qu'on va chercher à avoir un résultat rapide plutôt qu'une structure solide, afin de convaincre un éditeur de nous prendre sous son aile.

Cependant, une fois ceci fait, si on ne revient pas en arrière pour reprendre avec une procédure structurée, le projet acquiert de plus en plus d'inertie, le coût de rétro-ingénierie ne cesse de croître, et on finit par ne plus pouvoir maintenir proprement le logiciel quand bien même la volonté serait là.

Je pense que ce problème affecte le logiciel étudié, du fait que toute l'équipe est consciente du manque de structure et de documentation ainsi que des problèmes que cela pose, mais sans pour autant qu'une solution soit mise en place. Au vu des délais de plus en plus courts que nous observons et de la taille actuelle du code, il m'apparaît donc impensable de faire marche arrière à ce stade.

En conclusion, j'aurai donc pu me rendre compte par l'exemple à quel point il est important de ne pas négliger la formalisation et la structuration d'un logiciel, et ce du début jusqu'à la fin de son cycle de vie, ainsi que les problèmes qu'un manquement à cette règle peut engendrer.

*B.2.8 Semaine 8 (*20/09/2010 – 26/09/2010*)*

Cette semaine fut particulière dans le sens où il nous a été demandé de faire des heures supplémentaires, le jeu devant sortir en France le jeudi de la semaine suivante (le 30). J'ai donc travaillé également le samedi et le dimanche, ce qui a fait une semaine de travail de 7 jours.

Toujours attelé à la tâche de correction de bugs, ceux-ci sont mon quotidien. Cela peut parfois être démoralisant de voir que la vitesse à laquelle on m'assigne de nouveaux bugs est la même que celle que je prends à en résoudre, ce qui donne l'impression que le travail n'avance pas et/ou ne sera jamais terminé. Cependant le cœur du logiciel est stable, et la majeure partie des bugs ne sont pas "bloquants", c'est-à-dire que bien qu'ils soient sources d'erreurs, le jeu peut quand même continuer normalement.

N'ayant pas grand-chose de neuf à exprimer cette semaine, je vais m'attarder un peu plus sur le fonctionnement de ces fameuses "bases", car j'ai eu à travailler régulièrement avec elles cette semaine.

Tout d'abord, nous retrouvons la base des interfaces, qui contient le nom de tous les boutons, menus, titres, etc. Celle-ci est structurée en séries de "Labels" qui sont des identifiants uniques délimitant des groupes de noms. Chaque ligne de ces groupes contient donc un nom, traduit dans toutes les langues du jeu – une colonne correspondant à une langue. Pour obtenir un nom dans le code, nous faisons appel à une fonction getter qui, sur base du Label, du numéro de ligne depuis le Label et de la langue, va récupérer la cellule correspondante dans la base. Il est donc aisé de rajouter/modifier un titre et de le faire traduire.

Ensuite vient la base des actions, dont chaque ligne contient les définitions des actions spécifiques que l'on peut entreprendre dans le jeu, de la demande l'autorisation d'employer l'arme nucléaire à l'augmentation des effectifs dans la santé. Chaque action est identifiée par un entier, qui est également définit dans le code comme une constante. La ligne contient l'impact sur la popularité du joueur de l'action en question dans les colonnes représentants les différents "points sensibles", comme le libéralisme, la démocratie ou encore la sécurité. Elle contient aussi éventuellement un lien vers une réaction à lancer dans la base des réactions, et/ou un code à envoyer à une fonction qui va gérer des impacts autre que populaires.

La base des réactions est structurée en “pools” de réactions. Chaque réaction possède un identifiant unique sous forme d’un entier, et appartient à un pool définit par un label unique. Chaque réaction possède un certain nombre de paramètres typés, ainsi qu’une gravité. Cette gravité va définir quelle réaction choisir dans un pool ; par défaut, on prend celle qui possède la gravité définie en paramètre de l’appel, mais il existe des constantes que l’on peut spécifier pour demander un comportement différent, comme ALL qui va demander à ce que toutes les réactions d’un pool soient choisies les unes après les autres.

Ce mode est pratique pour les réactions qui comportent un test qui les empêche de sortir s’il n’est pas respecté. Ainsi, on peut définir un pool de réaction identifié par un label unique, dont la gravité est ALL et les tests mutuellement exclusifs. La base des réactions choisira donc la réaction appropriée d’elle-même sur base d’un et d’un seul appel.

Chaque réaction peut, comme les actions, envoyer un code à une fonction particulière pour un impact spécial, et peut également appeler d’autres réactions. Ainsi par exemple, la réaction qui correspond à une diminution des effectifs de la santé peut appeler des réactions correspondantes à des personnages manifestant leur mécontentement, voire même déclencher des grèves. Eventuellement, une réaction peut avoir un texte, traduit également dans toutes les langues du jeu. Ce texte correspond à ce qui sera affiché dans le jeu si la réaction est celle d’un personnage dans le jeu.

Comme dit précédemment, il existe d’autres bases, comme la base des nations et des organisations, mais qui contiennent principalement des données “inertes”, i.e. utilisées pour des précalculs qui seront enregistrés dans des fichiers .dat lus en début de partie pour initialiser les tableaux et variables des nations, personnages, organisations, etc. On retrouve par exemple les noms des villes, pays, personnages, etc., les informations sur l’économie des nations comme leur déficit, leur croissance, leur inflation, etc., l’intervalle entre les sessions de l’ONU, ce à quoi sont sensibles les personnages et associations (l’écologie, le libéralisme, etc.), et toute autre donnée inerte servant de “base” au jeu.

*B.2.9 Semaine 9 (*27/09/2010 – 01/10/2010*)*

Cette semaine fut la semaine de "rush" pour corriger le maximum de problèmes avant la sortie prévue le jeudi 30. Avec en parallèle le test des producteurs, nous nous voyons assigner des tâches autres que du simple debugging, c'est-à-dire des tâches qui impactent plutôt le Game Design. Par exemple, changer l'impact d'une loi ou d'une action sur le peuple. Ces éléments sont à la base tout à fait fonctionnels dans le code, mais ont un impact négatif sur le gameplay, et c'est ce que nous devons corriger dans ce cas. Ces tâches sont parfois beaucoup plus lourdes que du debugging car elles nécessitent quelques fois des connaissances beaucoup plus poussées du code, ou simplement demandent beaucoup de tests et d'ajustements par essai-erreur.

Je profite de cette entrée pour faire le point sur les différentes IA ou pseudo-IA que j'ai pu identifier dans le jeu. Tout d'abord, il y a les unités dans le "Wargame". Quand on donne l'ordre à un tank de se déplacer d'un point A à un point B, c'est un programme de pathfinding qui va se charger de tracer son chemin. La décision d'attaquer est purement réactive : quand une unité ennemie croise son rayon d'action, l'unité va s'arrêter pour l'attaquer. Le résultat n'est autre que de l'application de fonctions.

Ensuite, il y a le peuple, les personnalités et les associations, qui réagissent aux actions du joueur. Ils possèdent tous des critères qui définit ce qu'ils aiment et ce qu'ils n'aiment pas, et chaque action peut référencer ces différents critères. Ainsi, mener une politique d'intégration va mécontenter les groupes extrémistes mais réjouir les défenseurs des droits de l'homme. Cela va se traduire par une variation de popularité du président, et dans le cas des personnalités/associations, par une variation de leur fidélité au président. Si leur fidélité est basse, ils sont mécontents et peuvent par exemple lancer des mouvements de grèves ou des manifestations, ou dans le cas d'un ministre, démissionner ou médire dans la presse.

Les chefs d'état des nations étrangères ont également une IA propre, mais encore une fois réactive. A la manière des personnalités intérieures, leur "fidélité" par rapport aux autres chefs va varier selon leurs actions. Envoyer une aide financière lors d'une catastrophe est bénéfique, envoyer ses troupes en manœuvres aux frontières mécontentera. En mode simulation mondiale, l'IA n'est jamais pro-active : elle se contente de réagir aux actions du joueur. Ce n'est pas le cas en mode compétition où les chefs d'état des 16 pays jouables sont très actifs, tel que je l'ai décrit dans la semaine 5.

Un dernier élément est celui des négociations de contrats économiques ou militaires entre pays. Une IA va toujours chercher à maximiser son profit tout en prenant soin de garder un profit pour le joueur. Typiquement, elle va chercher à identifier les nombres qui lui apportent le plus d'utilité, et ceux-ci définis, elle calculera l'intérêt du joueur, de sorte à ce que celui-ci soit toujours au-dessus d'un seuil acceptable mais inférieur ou égal à celui de l'IA.

On remarque donc beaucoup de petites IA décentralisées, largement réactives, sauf en compétition où elles sont pro-actives. Le système est assez intéressant en tant que tel, donnant beaucoup de facilité à définir de nouvelles actions ou réactions, vu que l'impact sera automatiquement défini si on calcule bien nos critères lors de la création. Il eut été cependant peut-être intéressant de donner plus d'intelligence aux chefs d'états étrangers dans le mode simulation mondiale, pour dynamiser un peu plus le jeu.

B.2.10 Semaine 10 (04/10/2010 - 07/10/2010)

Cette semaine fut surtout centrée sur la résolution d'un mauvais calcul pour le taux d'espérance de vie et de mortalité, qui variaient trop fortement au cours du temps. C'est ici que j'ai pu avoir un meilleur coup d'oeil sur une partie du fonctionnement du moteur économique du jeu. Ce dernier illustre bien les problèmes que l'on peut rencontrer dans un processus non suffisamment documenté : bien qu'il soit dans l'ensemble bien réalisé et bien pensé, le moteur éco n'a été développé que par une seule personne, aujourd'hui absente de l'entreprise. Au vu de sa taille, de sa complexité et de l'absence de documentation, il est aisé de comprendre la grande réticence des programmeurs à aller y travailler.

Le principal problème que j'ai eu à corriger ce taux est qu'il était défini par un calcul complexe prenant en compte plusieurs autres taux qui, eux aussi, étaient définis par d'autres calculs complexes, et ainsi de suite. Ces calculs n'ayant aucune documentation, il n'était pas aisé de comprendre pourquoi on utilisait telle variable et de telle manière : il y avait-il là une structure bien pensée, ou était-ce le résultat d'ajustements successifs par essai-erreur ?

Finalement, après avoir essayé moult solutions pour diminuer la sensibilité du calcul, j'ai simplement décidé d'adoucir directement la variation du taux en agissant sur le delta entre la nouvelle valeur et l'ancienne, de sorte à ce que l'impact du calcul trop sensible prenne beaucoup plus de temps à se faire sentir sur le taux. Solution un peu bricolage s'il en est, mais qui a permis de résoudre le problème tout en gardant la logique du taux.

B.2.11 Semaine 11 (11/10/2010 - 14/10/2010)

Deux semaines après la sortie du jeu, j'ai pu acquérir une motivation nouvelle. En effet, si au départ le fait que le produit parvenu aux joueurs comprenait encore un grand nombre de bugs était assez démoralisant, à présent cela me motive d'autant plus à résoudre ceux-ci le plus rapidement possible pour apporter plus de satisfaction aux joueurs.

Au-delà de simplement corriger les disfonctionnements, j'essaie à présent de plus en plus à apporter des améliorations quand cela est possible. En plus de briser la routine de la correction de bug et d'apporter plus de créativité dans mon travail, cela me permet d'avoir plus de rapports avec les Game Designers, et donc d'enrichir ma vision du produit avec des points de vue plus haut niveau.

Cette manière plus détendue et plus orientée utilisateur de travailler m'a aidé à regagner confiance et motivation dans le développement, après ces dernières semaines qui furent un peu difficiles sur ces points.

B.2.12 Semaine 12 (18/10/2010 - 22/10/2010)

Cette semaine fut riche en définition et implémentation d'améliorations, avec en particulier la mise en évidence des articles de journaux qui sont responsables d'une variation de popularité, et le recalcul des probabilités du système d'assassinat. En effet, j'ai eu l'occasion de clôturer la correction des bugs importants qui m'avaient été attribués, ce qui fait que j'ai maintenant beaucoup plus l'occasion d'apporter des ajouts de fonctionnalités, plutôt que de simplement réparer l'existant.

Nous sortons toujours des patches à intervalles réguliers, et nous amenons petit à petit le jeu à un niveau de stabilité raisonnable, ce qui est pour le moins encourageant. Après avoir terminé plus de deux tiers des tâches qui m'avaient été attribuées début septembre, j'ai de nouveau l'impression de fort bien avancer, vu que les rapports de bugs arrivent moins vite que la correction des anciens. J'espère pouvoir bientôt passer à des tâches de développement à part entière.

B.2.13 Semaine 13 (25/10/2010 - 28/10/2010)

Cette semaine fut principalement axée sur la réintégration de la rubrique monnaie et des actions et données qui lui sont liées. Tout d'abord, il m'a fallu réactiver la rubrique dans le code et la base des actions, et ensuite faire le tour des actions pour vérifier qu'elles fonctionnaient correctement. En premier lieu, j'ai corrigé les actions d'entrée et de sortie de l'euro, avec surtout des ajustements de l'impact sur la popularité, de délai et de coordination avec la rubrique de l'Union Européenne, ainsi que l'ajout de la nouvelle devise et le changement de la devise affichée au profit de celle-ci lorsqu'on quitte l'euro.

Ensuite, il m'a fallu tester et analyser en profondeur l'impact du taux directeur sur le jeu. Celui-ci doit influer sur l'inflation, la croissance et la valeur de la devise, ce qui impacte donc le PIB et le déficit budgétaire. Pour m'aider dans cette tâche, j'ai implémenté une fonction qui récolte automatiquement toute une série de données sur des nations clés, ce qui m'a permis de faire tourner plusieurs parties avec des paramètres de test bien définis pour ensuite récupérer les données et les analyser.

Au final, j'ai pu constater que le taux directeur fonctionnait correctement, après un ajustement de ma part au début du test.

B.2.14 Semaine 14 (02/11/2010 - 04/11/2010)

Cette courte semaine fut consacrée à l'intégration de quelques modifications et corrections suite à la réintroduction des devises, telles qu'une fenêtre de confirmation demandant au joueur s'il veut suivre la nouvelle devise de sa nation à l'affichage, ou encore la correction de la conversion des dons entre nations.

Tout ceci fut entrecoupé de corrections de bugs relativement mineurs, pour se "remettre" de la grosse modification qu'a été la rubrique des monnaies.

B.2.15 Semaine 15 (08/11/2010 - 10/11/2010)

Deuxième semaine de corrections et modifications suite à la réintroduction des devises. Cette fois-ci avec des changements plus importants : les historiques. Ceux-ci sauvegardent chaque mois près d'une centaine de variables pour toutes les nations, telles que le PIB, la croissance, le taux directeur, ou encore la densité de population et le taux de natalité. Elles sont strictement réservées au joueur, qui peut les visionner à l'aide d'un graphique.

Le problème ici était que certaines de ces valeurs n'étaient pas enregistrées dans la devise interne (le "dollar moteur"). De ce fait, elles devenaient incohérentes une fois que la nation changeait de devise. J'ai donc implémenté une solution pour que tous les montants économiques soient enregistrés en devise interne, et j'ai rajouté une colonne d'options dans la base des historiques pour permettre de spécifier quelles variables on doit convertir en devise affichée lorsqu'elles sont visionnées.

Ensuite, j'ai implémenté une solution pour mettre à jour les sauvegardes obsolètes, car en parallèle de nos travaux, les joueurs sont également actifs. Il serait très dommageable que leurs sauvegardes deviennent incohérentes après un ou plusieurs patches. Cela ne fut pas une mince affaire car les changements étaient conséquents, mais c'est une habitude que je veux conserver, dans l'intérêt des joueurs.

B.2.16 Semaine 16 (15/11/2010 - 19/11/2010)

Cette semaine vit la correction d'un élément assez fondamental et qui peut s'avérer très confus : la conversion entre les devises. En effet, après mes travaux sur la rubrique des monnaies, je me suis rendu compte d'erreurs dans les méthodes de conversion pour les transactions d'un pays à l'autre, et les conversions entre devise interne et devise à l'affichage.

Le premier cas fut assez vite corrigé, vu qu'il s'agissait d'une simple inversion : pour convertir un montant d'une nation à une autre, on multipliait le montant par le rapport entre la valeur de la devise réceptrice et la valeur de la devise donatrice. Cela était erroné, du fait que si la valeur de la devise donatrice augmente et que le montant reste fixe, la nation réceptrice aurait reçu un montant moins important.

Le deuxième cas fut des plus compliqués. Les montants économiques dans le jeu sont d'abord calculés en devise interne, le dollar moteur, dont la valeur est constante. Toutes les devises "jouables" comme l'euro ou le "vrai" dollar ont un taux de conversion constant avec le dollar moteur. Ce taux est utilisé lorsque l'on veut afficher un montant interne dans la devise à l'affichage correspondante. Ces devises ont également une valeur, égale à 1 en début de partie, et qui évolue avec le temps. La multiplication du taux de base par cette valeur donne le taux courant, qui est affiché au joueur.

Deux taux cohabitent donc : le taux de base, constant, utilisé pour afficher les montants dans la devise de la nation ; et le taux courant, affiché au joueur et utilisé pour les transactions entre différentes devises jouables. Le taux de base est nécessaire pour que les montants affichés au joueur dans la devise de sa nation ne varient pas avec le temps - un billet de 10€ reste un billet de 10€, peu importe sa valeur.

Un problème se posait dans deux situations. La première est quand la devise à l'affichage est différente de celle de la nation - par exemple, prendre le Royaume-Uni et afficher les montants en euro. En ce cas, la conversion était erronée, car on ne considérait que la devise à l'affichage ; donc, les montants affichés ne variaient pas en fonction du taux courant de la devise de la nation et de la devise à l'affichage, ce qui est incohérent. J'ai donc corrigé la conversion pour l'ajuster en fonction du rapport des valeurs des devises.

La deuxième situation problématique était quand la nation du joueur changeait de devise - par exemple, quand elle passait à l'euro. La confusion causée par le nombre de taux et devises en jeu n'aidant pas, j'ai d'abord imaginé qu'il fallait effectuer une conversion supplémentaire, de devise initiale à devise courante, avant devise courante vers devise à l'affichage. Ceci était incorrect, car cela revenait à ne rien changer (la conversion étant transitive, convertir A vers B puis B vers C revient à convertir A vers C, et donc ignorer la devise courante).

Cette tâche m'occupa pendant plusieurs journées, mais aurait pu s'effectuer plus rapidement si j'avais fait montre de plus de rigueur dès le départ, m'évitant ainsi d'avoir l'esprit confus par nombre de variables s'intersectant.

B.2.17 Semaine 17 (22/11/2010 - 26/11/2010)

Parmi quelques corrections de problèmes mineurs, cette semaine vit surtout la correction de deux éléments économiques assez importants : la balance commerciale et les allocations familiales.

La balance commerciale est calculée par la différence entre les exportations et les importations d'un pays. Comme beaucoup d'autres variables économiques, celles-ci sont "lissées", c'est-à-dire que pour éviter des variations trop sensibles des indicateurs, on "mélange" le résultat constaté de l'année précédente avec la prévision de l'année courante. En effet, chaque indicateur ne représentant pas un cumul est une prévision sur l'année comptable - en somme, le résultat que l'on devrait obtenir le 31 décembre si rien ne bouge. La méthode de lissage considère le temps écoulé depuis le premier janvier pour pondérer la somme du résultat précédent et de la prévision courante : le résultat précédent est pondéré par la part de l'année qui doit encore s'écouler, et la prévision par la part de l'année déjà écoulée. Ainsi, au début on considère surtout la continuation du résultat précédent, mais plus on avance dans l'année, plus on considère la prévision courante, jusqu'au 31 décembre où elle atteint sa précision maximale.

Le problème des imports/exports venait du fait que les valeurs de l'année de référence (2010, l'année précédant l'année de départ) étaient erronées. De fait, je me suis rendu compte que nombre d'indicateurs étaient initialisés deux fois : une première fois lors de la construction des bases, et une deuxième fois au lancement d'une partie. En enlevant tout recalcul d'indicateur déjà défini, le problème fut résolu.

Les allocations familiales, quant à elles, constituèrent un morceau plus difficile, du fait de la structure des variables de lois. En effet, chaque loi du jeu est liée à une structure, définissant par exemple sa valeur courante et sa valeur objective, quand elle fut votée pour la dernière fois et, le cas échéant, quand le prochain vote aura lieu.

Le problème vient du fait que cette structure ne comporte qu'une et une seule valeur pour définir sa valeur courante et objective, valeurs qui sont utilisées par le système de vote des lois pour définir les votes positifs et négatifs des différents partis, ainsi que l'impact de la loi sur les critères de fidélités et de popularité. Les allocations familiales étant composées de 3 valeurs (montant pour 1, 2 et 3 enfants), son implémentation d'origine fut de convertir les allocations en multiples de 10 et de les sauver de manière "brute" à l'adresse de la valeur de la loi dont on aura divisé les bits en 3. Cela causait nombre de problèmes, dûs à la conversion en multiples de 10 en plus de la conversion entre devise moteur et devise à l'affichage.

Pour rendre la gestion de cette loi plus propre, au lieu de sauvegarder les 3 valeurs dans une seule variable au sein de la loi, je les ai sauvegardées dans un tableau externe, avec leur moyenne en guise de valeur pour la loi. Les tableaux sont mis à jour en même temps que la loi, et le système de vote n'y "voit que du feu" vu que les tendances sont conservées (augmenter les allocations augmentera la moyenne, et vice-versa). Les méthodes de synchronisation ont été implémentées en gardant en tête que d'autres lois à plusieurs valeurs pourraient un jour avoir également besoin d'un tel traitement.

B.2.18 Semaine 18 (29/11/2010 - 03/12/2010)

La tâche récurrente de cette semaine fut d'ajuster la variation des devises à long terme. En effet, suite aux changements sur le taux directeur, certains pays finissaient par voir la valeur de leur devise exploser ou devenir très faible, ce qui causait de gros écarts de PIB vu que ceux-ci sont affichés après conversion dans la devise du joueur. Le but était donc de faire en sorte que l'IA ajuste son taux directeur d'une manière telle que le classement des PIB reste cohérent à long terme.

En premier lieu, j'ai cherché à borner la variation de la valeur des devises, en divisant les effets selon les indicateurs responsables (l'inflation, le taux directeur et l'épargne), et à borner l'impact de ces effets (par exemple, l'inflation ne peut pas faire varier la valeur de la devise au-delà ou au-deçà de 5 fois sa valeur). Cela fut implémenté la semaine précédente, mais n'était pas concluant - car même si les effets étaient limités, le classement restait incohérent.

La deuxième solution fut donc de faire varier le taux directeur. La question du comment se posait, et j'ai tout d'abord imaginé ajuster le taux directeur sur base de la croissance "prévue", c'est-à-dire la croissance que devrait afficher la nation pour atteindre la valeur de PIB prévisionnelle enregistrée dans les bases. L'essai ne fut pas concluant. J'ai donc ensuite cherché à calquer le taux directeur sur l'inflation - en somme, pour que les taux d'épargne suivent le degré d'inflation - mais cela ne fut toujours pas concluant. Finalement, j'ai tenté de calquer le taux directeur sur la croissance, et dans ce cas, le classement était probant.

Je cherche cependant d'autres méthodes, agissant seules ou combinées à d'autres, pour tenter d'obtenir un résultat encore meilleur, mais pour l'instant la méthode de la croissance reste la meilleure.

Une autre grande tâche cette semaine fut d'ajouter des unités aux panneaux récapitulatifs de l'économie, afin de les rendre plus compréhensibles. De par son ambigüité, il m'a fallu tout d'abord déterminer par lecture du code quelles unités donner à quels éléments, et ensuite modifier et harmoniser les méthodes de récupération des textes pour incorporer ces unités. Même si cela fut un peu fastidieux, cela a assez bien clarifié la lecture de ces panneaux, importants pour trouver quelle denrée vendre ou acheter et à qui.

B.2.19 Semaine 19 (06/12/2010 - 10/12/2010)

Au cours de cette semaine, j'ai pu conduire des tests supplémentaires sur la variation de la valeur des devises afin de raffiner le processus d'ajustement du taux directeur par l'IA. En explorant d'autres solutions, j'ai cherché à rajouter la prise en compte de l'évolution de la valeur par rapport à la valeur initiale, puis le statut du déficit budgétaire.

Le premier point a produit des résultats modérés ; après ajustement des bornes de variation, je l'ai tout de même conservé. Le second point par contre, produisait certains résultats forts incohérents – j'ai donc décidé de l'abandonner.

Ce calcul d'ajustement de taux directeur m'a donné matière à réflexion concernant une IA économiques pour les nations. En effet, en mode simulation aucune action n'est prévue pour que l'IA tente d'équilibrer son budget, ce qui implique un certain nombre d'incohérences à long terme, comme des déficits dépassants les 600% du PIB de la nation.

J'ai donc l'impression qu'il serait intéressant de développer une IA qui chercherait à équilibrer le budget d'une nation, en influant sur les divers paramètres économiques tels que le taux directeur et les budgets des ministères. Le nombre de variables à prendre en compte et la difficulté de prévoir les effets des actions – particulièrement à long terme – rend cependant pareille tâche fort ardue.

B.2.20 Semaine 20 (13/12/2010 - 17/12/2010)

Durant cette semaine, j'ai principalement travaillé sur la résolution de problèmes provenant des actions du joueur capturées par l'interface, telles que la demande d'une amnistie fiscale ou la demande de la levée de sanctions auprès de l'ONU, ainsi que du moteur économique, tels que des absences ou mauvaises utilisations de conversions entre devises.

Ayant effectué un certain nombre de tâches y ayant trait, je commence à aborder le moteur économique d'une manière moins rigide, plus à l'aide. En effet, celui-ci est considéré comme fort complexe et large, et le programmeur à l'origine de ce moteur ne fait malheureusement plus partie de l'équipe – ce qui implique une certaine lourdeur des démarches de debug et d'amélioration.

A propos du moteur économique, il est intéressant de remarquer qu'il en existe deux versions – le moteur dit non-contraint et le moteur dit contraint.

Le premier fut le concept original du moteur, où ce dernier ne ferait que proposer une structure permettant aux données renseignées de prime abord d'évoluer de manière naturelle. J'ai qualifié cette approche de bottom-up : l'évolution naturelle dicte les résultats nets (par exemple, les résultats des secteurs d'activité dictent le PIB d'un pays). Il échoua malheureusement du fait qu'il était très difficile à stabiliser et à maintenir cohérent du fait du nombre important de variables en jeu et de la longue durée d'activité.

C'est pour cela qu'une seconde version, le moteur contraint, vit le jour. Comme son nom l'indique, il adopte une autre approche, top-down cette fois-ci : les prévisions réalisées à priori dictent l'évolution des données (par exemple, sachant que le PIB est prévu d'augmenter de 10% d'ici à 2020, le résultat des activités doit augmenter globalement de 10%). De ce fait, il devient beaucoup plus aisé de contrôler et de stabiliser le moteur, vu qu'il va tenter de suivre les prévisions qu'on lui renseigne. Cela nous assure un minimum de cohérence à long terme.

B.2.21 Semaine 21 (20/12/2010 - 23/12/2010)

Cette courte semaine précédant Noël fut celle d'une importante mise en route : le développement du nouveau système de construction. En effet, durant les pauses il m'arrive de tester le jeu au travers de différents pays. Et j'ai remarqué un problème récurrent : celui des constructions souvent interminables. De fait, ne fut-ce qu'augmenter la production énergétique d'un pays de 50% pouvait prendre des millénaires.

Afin de pouvoir me lancer sur une tâche de développement un peu plus conséquente, j'ai donc proposé de porter des améliorations au système de construction. Après discussions avec le manager, une spécification du nouveau système a été fixée. Le changement majeur attendu est de porter la gestion des constructions d'un système séquentiel à un système en parallèle. De fait, le système actuel demande qu'une construction soit terminée avant d'en lancer une autre d'un même type. Le nouveau système construirait tous les bâtiments en même temps.

Il faut cependant limiter ce parallélisme, pour éviter qu'il devienne lui-même irréaliste. L'idée d'introduire une limitation par la main d'œuvre disponible a donc vu le jour : on considère que chaque nation peut affecter un nombre maximal d'ouvriers qui dépend du personnel du secteur du bâtiment, et que chaque construction demande un certain nombre d'ouvriers. Toutes les constructions se feront donc en parallèle sous réserve qu'il y ait suffisamment de main d'œuvre pour s'occuper de tous les chantiers.

B.2.22 Semaine 22 (28/12/2010 - 30/12/2010)

Très courte semaine prise entre deux périodes fériées, qui fut mise à profit pour le développement du nouveau système de construction.

La semaine dernière vit l'implémentation des méthodes clés de calcul du temps et de la main d'œuvre requis pour les constructions, ainsi que la mise à jour de l'interface pour supporter le nouveau système. Cette semaine-ci fut donc plus orientée traitement interne, où j'ai parcouru les différentes méthodes de lancement de constructions afin de garder à jour la main d'œuvre disponible.

Un autre point important fut de garantir une fois de plus que les sauvegardes des anciennes versions puissent intégrer le nouveau système ; il a donc fallu trouver un moyen de récupérer toutes les constructions en cours afin de mettre à jour leur durée et de calculer la main d'œuvre occupée. Tâche ardue mais ô combien nécessaire.

B.2.23 Semaine 23 (03/01/2011 - 06/01/2011)

Troisième partie du développement du nouveau système de construction, encore une fois au sein d'une courte semaine (à noter mon absence le 04, principalement pour cause de fatigue).

Les principaux problèmes à ce stade furent liés aux structures d'évolution (les evols). En effet, chaque chantier est enregistré dans une structure d'évolution, qui prévient donc quand celui-ci se termine et quand la construction doit apparaître.

Le premier problème fut de conserver les données spécifiques aux chantiers, telles que la région, le coût, et l'éventuel bonus de main d'œuvre, car celles-ci sont nécessaires pour mettre à jour les données de la nation une fois le chantier terminé. Afin de rester aussi général que possible sans modifier trop profondément les structures d'évolution, il a été décidé d'utiliser un tableau de paramètres. Chaque evol se voit donc attribuer un indice de paramètre, facultatif. Cet indice permet de retrouver dans le tableau de paramètres pertinent pour l'evol la structure contenant les paramètres qui le concernent.

Le deuxième problème fut celui des constructions enchaînées. De fait, chaque région du jeu possède un encapsuleur de structure d'évolution (un Pevol) par type de bâtiment, qui peut contenir un et un seul evol. Donc, si on lance la construction d'un hôpital dans une région donnée, et que le lendemain on lance à nouveau une deuxième construction, l'evol lié au Pevol de la région devrait être mis à jour pour intégrer la nouvelle construction. Cependant, étant donné les dates de fin distinctes des constructions, cela n'est pas possible – l'evol n'a qu'une et une seule date de fin.

Pour résoudre ce problème, il a donc été décidé de chaîner les evols de construction, de sorte que chaque Pevol contienne le premier evol de la liste, qui peut alors pointer vers un evol supplémentaire, et ainsi de suite. Cette liste est triée par ordre décroissant de date de fin, de sorte que l'evol se terminant le plus tard soit le premier.

B.2.24 Semaine 24 (10/01/2011 - 14/01/2011)

Cette semaine fit le pont entre le développement du nouveau système de construction et le lancement d'une fonctionnalité toute à fait nouvelle et plus complexe : la nationalisation et la privatisation de secteurs d'activité.

Cette fonctionnalité permet donc à un état de racheter des parts d'un secteur d'activité afin de récupérer son résultat et d'en contrôler certains aspects, ou de revendre les parts d'un secteur qu'il détient afin d'obtenir un input financier important et rapide.

Ce système est prévu à long terme – i.e. pour la version du jeu qui sortira dans un peu moins d'un an. Vu qu'il touche profondément au moteur économique et que j'ai pu passer un certain temps à travailler sur ce dernier, cette tâche de développement m'a donc été confiée.

B.2.25 Semaine 25 (17/01/2011 - 21/01/2011)

Deuxième semaine de développement pour le système de privatisation/nationalisation (que j'ai baptisé Privatinat pour les besoins). Celui-ci se déroule assez bien, alternant les phases de discussion avec le management afin de fixer des détails de spécification ou de résoudre des problèmes nouveaux apparus lors de l'implémentation, avec les phases de coding actif.

Après avoir implémenté les premières méthodes, le besoin d'une interface de test se fit rapidement sentir. Désireux d'avoir un système propre et surtout testable facilement par d'autres personnes, j'ai pris la peine de développer une interface plus ou moins propre, tout en gardant en tête que la structure d'interface du jeu peut changer d'ici la sortie de la fonctionnalité.

Afin d'implémenter les spécifications du système, j'ai opté pour une approche cherchant à maximiser l'encapsulation – le moteur économique étant une partie assez black-box du code dans la culture de l'entreprise, j'ai préféré faire en sorte que mon implémentation puisse également être utilisée de manière black-box. A cet effet, la majeure partie des méthodes ont été implémentées dans la structure des activités même, afin que l'on puisse manipuler les informations facilement à l'aide de getters et setters. De même, j'ai accordé de l'importance à étoffer la documentation des fonctions, afin qu'elles puissent être utilisées et comprises sans qu'il soit nécessaire de s'attarder sur leur implémentation.

Le développement de cette fonctionnalité a cependant une pression assez importante : le temps. En effet, mon stage se terminant la semaine prochaine, il est impératif que le cœur du système soit en place et soit surtout solide.

B.2.26 Semaine 26 (24/01/2011 - 28/01/2011)

Et voici donc passée ma dernière semaine de stage. Elle fut consacrée à la finalisation de Privatinat, pour lequel j'ai rédigé et suivi des tests de cas réguliers et extrêmes. Ceux-ci ont permis de capturer un certain nombre d'erreurs et de mettre en évidence certaines lacunes du système, que j'ai donc pu corriger à temps.

Au milieu de la semaine, avec un peu de temps supplémentaire sur les bras, j'ai pu me mettre à la rédaction de réactions de la part de l'IA afin d'aiguiller le joueur dans la gestion des secteurs nationalisés.

Il restera cependant quelques améliorations à apporter au système, et surtout plusieurs pistes ouvertes d'amélioration du moteur économique en général, que j'ai pris soin de noter et de détailler pour du futur travail.

Voilà donc ce stage clôturé, et bien que son objet ne fût pas celui escompté, son expérience reste fort pertinente et enrichissante. Le personnel me laisse également une forte impression, de telle sorte que le départ du bureau le dernier soir ne se fit pas sans peine.